VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačního a znalostního inženýrství

Doplňková validace HTML a XHTML dokumentů

Bakalářská práce

Petr Nálevka

Vedoucí práce: Ing. Jiří Kosek

květen 2005

Anotace

Cílem této práce je vytvoření nástroje pro validaci dokumentů značkovaných v jazyce HTML. Takovýto nástroj jde za hranice běžně používané validace oproti DTD definicím. Díky využití moderních validačních jazyků, je možné formalizovat dodatečná omezení vycházející ze specifikací jazyka HTML, které přitom není možné vyjádřit s využitím jazyka DTD. Takovýto validační nástroj pak může pomoci autorům HTML dokumentů v lepším dodržování obecně uznávaných standardů, a tím může přispět ke zlepšení celkové dostupnosti těchto dokumentů a k použitelnosti této technologie vůbec.

První kapitola této práce se zabývá jednotlivými standardy a doporučeními, které vymezují používání jazyka HTML a kladou na něj některá zásadní omezení. Cílem této části je základní zmapování oblasti standardizace jazyka HTML a rozbor jednotlivých typů omezení, která jsou na tento jazyk jednotlivými standardy kladena. Takováto analýza pak slouží jako výchozí bod pro kapitolu druhou.

Cílem druhé kapitoly této práce je výběr několika v praxi používaných validačních jazyků a posouzení jejich vhodnosti pro popis dodatečných omezení vycházejících ze standardů HTML. Výsledkem je pak volba konkrétního jazyka nebo kombinace jazyků, která je dále použita pro formalizaci těchto omezení.

Třetí kapitola popisuje implementaci dodatečných omezení ve vhodném validačním jazyce. Zabývá se tím, jaké přístupy a jaká architektura byla zvolena a jaké jsou její slabé a silné stránky. Popisuje strukturu a modularitu zvoleného formálního zápisu a metodiku jeho implementace.

Čtvrtá kapitola detailně popisuje implementaci a architekturu prototypové aplikace pro validaci HTML dokumentů, která využívá dříve definované formalizace. Zabývá se nejen použitou technologií, ale i uspořádáním jednotlivých komponent a podporovanou funkcionalitou. Posuzuje i využitelnost této aplikace v praxi.

Annotation

The aim of this thesis is to create a tool able to validate HTML documents. Such a tool goes beyond the limits of the commonly used DTD-based validation. Additional restrictions specified in the HTML standards may be expressed, thanks to modern advanced validation languages. Such a validation tool may then help authors of HTML documents adhere better to commonly recognized standards and thereby improve the accessibility of those documents and the usability of this technology in general.

The first chapter of this thesis describes various HTML standards and recommendations and some fundamental restrictions of this language defined in those standards. This chapter analyses the scope of HTML standardization and various types of restrictions involved in those standards. Such analysis will serve as a starting point for the second chapter

The objective of the second chapter is to select an eligible subset of commonly used validation languages. Different characteristics of those languages are considered to choose the right language or combination of languages suitable for formalization of additional restrictions resulting from the HTML standards.

The third chapter describes the particular implementation of the aforementioned restrictions using the chosen validation solution. This chapter discusses the selected approaches, their architecture and relative pros and cons. It describes the modularity of the definition as well as the implementation methodology.

The last fourth chapter is a detailed description of the implementation and architecture of a prototype HTML validating application, which makes use of the validation definitions that have been formely defined. It deals with the used technology as well as different organization of the different components and supported functionality. It also investigates the practical utilization of the application.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a použil pouze literaturu uvedenou v přiloženém seznamu. Nemám námitek proti půjčení práce se souhlasem katedry ani proti zveřejnění práce nebo její části.

V Praze dne 6. května 2005

Petr Nálevka

Obsah

Úvod

S dramatickým rozvojem Internetu v posledním desetiletí rostla stejně dramaticky i obliba jazyka HTML, který se stal de facto standardem pro publikování elektronických dokumentů na této síti. Tak jak je stále více informací publikováno právě v tomto formátu, roste i přirozená potřeba zpřístupnit tyto dokumenty co nejširšímu okruhu uživatelů. HTML tedy musí umožnit autorům vytvářet dokumenty v libovolném přirozeném jazyce tak, aby byly přístupné uživatelům z různých zemí světa. Důležitou otázkou je jistě i dostupnost pro handicapované uživatele. Například pro nevidomé je typická grafická interpretace dokumentů vyloučena. V poslední době výrazně roste na významu také potřeba dostupnosti z alternativních platforem (hardware/operační systém). Toto je pravděpodobně dáno vzrůstající diverzifikací aplikačního prostředí na pracovních stanicích, které jsou právě typicky využívány pro přístup k HTML dokumentům, a také možnostmi přístupu z mobilních zařízení, jakými jsou například mobilní telefony nebo PDA. Tato zařízení jsou vzhledem k omezeným systémovým zdrojům výrazně citlivá na správné dodržování syntaxe jazyka HTML.

Zmíněné trendy vedou nejen k rostoucí potřebě standardizace jazyka HTML, ale hlavně vyvíjí tlak na dostupnost efektivních nástrojů, které mají napomoci dodržování standardu v praxi. Teprve s dostupnými a široce použitelnými validačními nástroji je možné efektivně podporovat autory HTML stránek v dodržování standardů, a zlepšovat tak dostupnost těchto dokumentů nezávisle na použité technologii, jazyku a schopnostech uživatelů.

Za validační nástroj přitom v žádném případě není možné považovat internetový prohlížeč schopný zobrazovat HTML dokumenty (dále jen prohlížeč). Ten totiž nemá standardně definovaný způsob ošetření chyb viz. [HTML4]. Většina všeobecně rozšířených prohlížečů se snaží nějakým způsobem zobrazovat i dokumenty, které nevyhovují standardu, a to dokonce i po stránce struktury dokumentu a nebo základní syntaxe značek. Jde přitom o logický krok. Měřítkem použitelnosti daného prohlížeče pro běžného uživatele jistě bude jeho schopnost zobrazit co nejvíce různých dokumentů v čitelné podobě. V praxi je tedy internetový prohlížeč vhodný pro odladění dokumentů pouze v rámci vizuálního uspořádání. Pro tvorbu dokumentů splňujících standardizační specifikace je nezbytně nutné validovat příslušné dokumenty oproti těmto standardům. S validací mohou výrazně pomoci právě automatické validační nástroje. Ty sice většinou nedokáží kontrolovat všechna omezení, ale s rostoucími schopnostmi a komplexností validačních jazyků, jsou jich schopny popsat stále více. Vynechání validace, jakožto součásti publikačního procesu, může vést k produkci nestandardních dokumentů, které jsou čitelné pouze pro úzký okruh masově používaných prohlížečů.

Tímto mechanizmem zde postupně vzniká jakýsi začarovaný kruh. Kromě standardního HTML jazyka existuje ještě jakýsi pseudo-jazyk, jehož podpora v prohlížečích byla vynucena z historických důvodů a ze snahy zpřístupnit autorům dokumentů některé nadstandardní funkce, které zvyšují atraktivitu daného prohlížeče oproti konku-

renci. Podpora takového pseudo-jazyka je stále nutnou součástí každého prohlížeče, který si klade za cíl zpřístupnit maximální množství dokumentů. Takováto podpora je pak velkou zátěží pro vývoj nových použitelných prohlížečů. Výrazně ztěžuje například implementaci odlehčených verzí pro mobilní zařízení a obecně se dá říci, že zhoršuje ve všech směrech dostupnost HTML dokumentů. Zhoršuje také konkurenci na tomto poli, a výrazně tak zpomaluje implementaci nových standardů pro tvorbu a zobrazování HTML dokumentů, což vede k ustrnutí celé oblasti na mrtvém bodě.

Zmíněné nepříznivé vlivy zasahují přitom celou oblast informačních technologií. V případě HTML nejde dnes jen o dokumenty v klasickém slova smyslu. Dynamická podoba HTML se v podstatě stala hlavním prostředkem pro tvorbu distribuovaných uživatelských rozhraní pro Klient/Server aplikace ve firemním, ale i domácím a státním prostředí. Prohlížeč se tak stává jakýmsi univerzálním klientem pro přístup k libovolným datovým zdrojům a službám. V tomto kontextu je důležitá multiplatformnost, jako jedna z klíčových výhod tohoto řešení. Data i aplikační logika pak může být dostupná z libovolného terminálu, třeba i z mobilního zařízení, bez potřeby dalších implementačních úprav. Tato skutečnost potom pomalu spěje k naplnění známé vize IT, že „síť je počítač“ *(the network is the computer)*. Ovšem dosažení této vize pomocí HTML bude možné jen za předpokladu dostatečného prosazovaní standardizace tohoto jazyka v praxi.

Pomoci v tomto procesu se ostatně snaží i tato práce. Jejím cílem je vytvoření validačního nástroje, který by sloužil autorům HTML dokumentů k automatizovanému a jednoduše použitelnému odhalovaní nesouladů těchto dokumentů se standardy HTML. Oproti již existujícím aplikacím (například DTD validátor od W3C...), přináší nástroj popsaný v této práci validaci některých dodatečných omezení popsaných ve standardech jazyka HTML. Z historických a jiných důvodů je struktura jazyka HTML popsána v jazyce DTD *(Document Type Definition)*. K výhodám tohoto jazyka patří výborná čitelnost, stručnost a jednoduchost užívání. Ovšem jeho vyjadřovací schopnosti nedosahují zdaleka možností, jakými disponují moderní validační jazyky postavené na bázi XML. Jejich vyjadřovací schopnost je daleko za hranicí pouhého definování vzájemných strukturálních vazeb jednotlivých elementů, jak je známe z DTD. Pomocí těchto jazyků lze popsat i složitější omezení definovaná ve standardech, a poskytnout tak autorům jednoduchou cestu, jak dosáhnout většího souladu s nimi. Cílem této práce (hlavně kapitoly dvě) je provést analýzu možností dostupných jazyků a vybrat z nich jeden konkrétní nebo případně jejich kombinaci, která bude nejvhodnější pro formalizaci omezení kladených na HTML dokumenty. V další fázi budou vybrané validační jazyky použity k implementaci této formalizace tak, aby pouze rozšiřovaly omezení, která jsou již popsána stávajícími DTD definicemi. Bude tedy platit, že dokument, který je validní vůči těmto omezením, musí být pokud možno validní i proti oficiálním DTD definicím. Naopak to ovšem zdaleka platit nemusí.

V konečné fázi se tato práce zabývá návrhem a implementací prototypové validační aplikace, která umožňuje uživatelům automatickou validaci dokumentů prostřednictvím webového rozhraní. Poskytuje jim maximum informací o nalezených chybách, včetně čísla řádky a popisu problému, s možným nastíněním postupu k jeho odstranění. Některá testovaná omezení jsou volitelná ze strany uživatele. Je tak možné zvolit si stupeň striktnosti a vybrat skupiny omezení, které mají být zahrnuty do validačního procesu.

Kapitola 1

Analýza omezení a požadavků kladených na HTML a XHTML dokumenty

1.1. Úvod

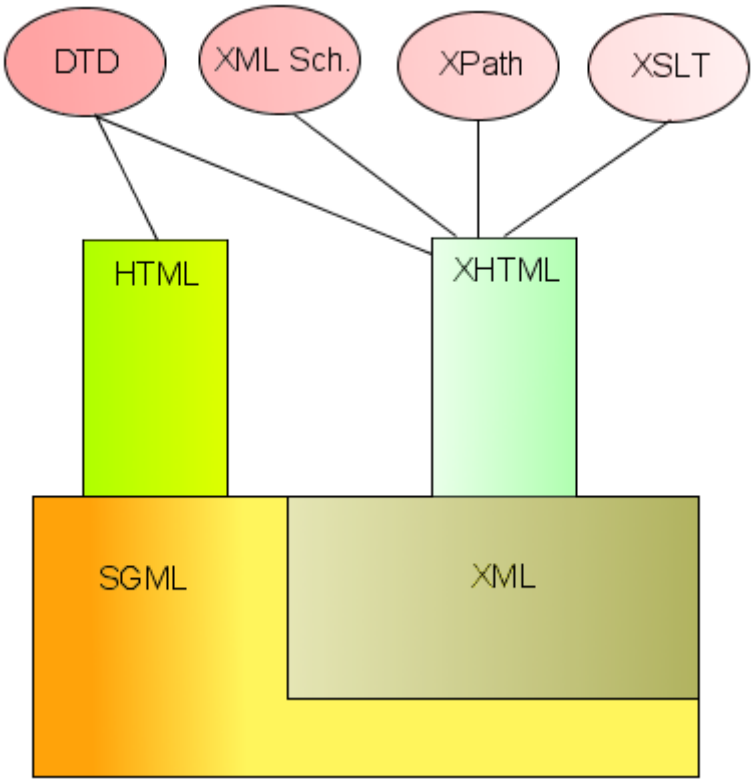
Existuje několik zdrojů omezení, která jsou kladena na dokumenty v jazyce HTML. Hlavním zdrojem standardizačních aktivit je v dnešní době W3C (*World Wide Web Consorcium*), které vydává tzv. doporučení v podobě specifikací jazyka HTML. Standardizací HTML se zabývá i Mezinárodní organizace pro standardizaci (ISO) spolu s Mezinárodní elektrotechnickou komisí (IEC). Jejich specifikace ISO/IEC 15445 viz. [ISO15445], známá též jako ISO-HTML, vychází z HTML 4.01 od W3C. ISO-HTML je v podstatě podmnožinou HTML 4.01, neboť tuto specifikaci dále omezuje, ale nerozšiřuje. Dá se tedy říci, že dokument, který je validní podle ISO-HTML, bude validní i podle specifikace W3C (nemusí platit obráceně). Kromě zmíněných standardů existuje ještě celá řada dalších, které se snaží nějakým způsobem vést autory k psaní kvalitnějších dokumentů a vývojáře prohlížečů k jejich lepšímu zobrazování. Jedním z nich je i směrnice WCAG, která se snaží o zpřístupnění HTML stránek co nejširšímu okruhu uživatelů. Touto směrnicí se budeme také dále zabývat.

1.2. Doporučení W3C

Doporučení W3C definují celou škálu variant jazyka HTML. Pokud pomineme starší verze, které není doporučeno používat, je v dnešní době aktuální HTML 4.01, které je díky dobré kompatibilitě s prohlížeči velice rozšířené. Z tohoto jazyka dále vychází jazyk XHTML 1.0, který není v podstatě ničím jiným než restrikcí HTML 4.01 tak, aby splňoval omezení jazyka XML. To přináší celou řadu výhod. Zpracování takového dokumentu je o poznání jednodušší. Navíc lze využít bohatou škálu nástrojů vyvinutých pro jazyk XML, jako jsou různé parsery, transformační nástroje nebo validační jazyky. Další verze XHTML pak přináší modularizaci jednotlivých konstruktů jazyka, což umožňuje vytvářet libovolné podmnožiny jazyka HTML podle potřeby. V této práci se zaměříme hlavně na XHTML 1.0 a HTML 4.01. Důvodem je jejich široké používání v praxi. Také platí, že HTML 4.01 je v podstatě poslední verzí, kde došlo ke změně množiny elementů a atributů jazyka. Další verze postavené na XML totiž samotný jazyk nijak závažně nemění. Specifické modifikace jazyka, jako je například XHTML Print, XHTML Basic a další, jsou nad rámec této práce.

Doporučení W3C rozeznávají v zásadě dva hlavní adresáty definovaných omezení. Jedni jsou lidé implementující prohlížeče HTML dokumentů. Ti musí vědět, jaký význam přikládat jednotlivým elementům jazyka HTML a jakým způsobem tyto elementy a jejich obsah interpretovat. Těchto omezení se bude tato práce dotýkat pouze okrajově. Už z principu nemohou být předmětem validace HTML dokumentů, a proto ani předmětem validace doplňkové. Druhou skupinou jsou autoři dokumentů. Omezení týkající se autorů budou dále podrobně rozebrána. Právě ta z nich, která nejsou podchycena v oficiálních DTD definicích, budou předmětem doplňkové validace.

Některá doporučení adresovaná autorům dokumentů se zabývají tím, jakým způsobem mají autoři ošetřit některé prezentační aspekty dokumentů ve stylových definicích. Tato omezení nebudou předmětem této práce, protože je není možné automaticky validovat na základě validace samotných HTML dokumentů.



Obrázek 1.1. Vztah SGML, XML, HTML a XHTML

1.3. HTML 4.01

1.3.1. Základní konstrukty jazyka SGML používané v HTML 4.01

Oproti starším verzím jazyka HTML přidává verze 4.01 viz. [HTML4] některé multimediální funkce, skriptování, podporu stylových definic, podporu tisku a lepší dostupnost dokumentů pro handicapované uživatele. Stejně jako dřívější verze je i tato odvozena z jazyka SGML *(Standard Generalized Markup Language)*.

SGML je v zásadě soubor pravidel, které by měly splňovat značkovací jazyky odvozené ze SGML. K definici odvozených jazyků se používá jazyk DTD. Ten vymezuje jednotlivé elementy (ty definují jednotlivé značky) a vazby mezi nimi. Takovými vazbami rozumíme například vymezení množiny subelementů pro daný element a jednoduchá pravidla jejich výskytu. Dále pak jednoduché definice obsahu elementů, jejich atributů a obsahu těchto atributů.

HTML umožňuje autorům značkovat dokumenty tak, aby vedle obsahu těchto dokumentů vyznačili ještě další strukturální, prezentační a sémantické informace. Základním konstruktem SGML je element. Ten vyznačuje v dokumentu určité struktury s příslušným chováním. Jako příklad můžeme uvést třeba element `<p>` označující odstavec a nebo elementy popisující tabulkové struktury. Typicky platí, že každá deklarace elementu vymezuje tři základní části. Je to počáteční značka, koncová značka a obsah. Některé elementy v HTML dovolují vynechat počáteční značku, a některé dokonce i koncovou. V tomto ohledu je HTML syntaxe postavená nad SGML velice volná. Některé elementy nemají obsah. Jde o elementy, u kterých by obsah neměl žádný význam. Příkladem je element `<br>`, který slouží pro označení zalomení řádky. Všechna tato pravidla jsou vyjádřena v DTD definici. SGML definuje i další pravidla pro elementy, například pravidla týkající se jejich vzájemného vnořování. Dále platí, že koncová značka ukončuje i všechny neukončené elementy až zpět k počáteční značce.

Příklad 1.1. Příklad nepovoleného vnořování elementů

```
<a> text <p> text </a> text </p>
```

Počáteční značka může obsahovat více atributů, které se zapisují za jménem elementu. Atributy se mohou vyskytovat v libovolném pořadí a mohou (nebo nemusí) mít nějakou hodnotu. Ovšem typicky platí, že atribut je dvojice jméno-hodnota. Hodnotu je možné uzavřít ve dvojici dvojitých, nebo jednoduchých uvozovek. Pokud obsahuje jen alfanumerické znaky a některá interpunkční znaménka, není ji nezbytně nutné takto ohraničovat. Dále platí, že na velikosti písmen ve jménech atributů ani elementů nezáleží.

Příklad 1.2. Příklad zápisu elementu s atributy

```
<a href="http://www.firmaxy.cz">
  <img src="/images/logo_xy.gif" width=150 height=55 alt="Logo firmy XY">
</a>
```

Některé znaky, které může být žádoucí zobrazovat v HTML dokumentu, mají specifický význam v rámci HTML/SGML. Abychom se vyhnuli konfliktům, je možné takové znaky vyjádřit jako numerickou, nebo symbolickou zkratku uvozenou znakem „&“ a ukončenou znakem „;“. Takovému vyjádření se říká znaková entita. Například < representuje znak „<“ a podobně. HTML definuje množinu znakových entit, které je možné používat v dokumentech. Kromě entit, které představují lidsky srozumitel-

nější možnost vkládání znaků, lze libovolný znak ze znakové sady používané dokumentem vyjádřit za pomoci číselné pozice tohoto znaku v této znakové sadě.

Krom běžných značek lze do dokumentů vkládat i komentáře viz. následující ukázka.

Příklad 1.3. Komentáře v HTML

```
<p>
  text
  <!-- komentář -->
</p>
```

Komentáře nemají mít žádný vliv na výsledné zobrazení. Po vykřičníku musí neprodleně následovat dvě pomlčky. Není zde povoleno žádné volné místo. Naopak po obou ukončovacích pomlčkách volné místo povoleno je. Uvnitř textu komentáře se nesmí vyskytovat další dvě a více pomlček za sebou.

1.3.2. Omezení deklarovaná v DTD

1.3.2.1. Elementy a atributy

Každý element jazyka HTML je společně se svými atributy deklarován v DTD. Tam je také popsáno jaké subelementy může daný element obsahovat. Pro tyto subelementy je možné vyznačit jednoduchá pravidla výskytu:

- element je povinný a vyskytuje se právě jednou,
- element je povinný a vyskytuje se jednou, nebo vícekrát
- element se vyskytuje jednou, nebo vůbec,
- element se vyskytuje jednou, vícekrát, nebo vůbec,
- element se může vyskytnout,
- element se nesmí vyskytnout,
- dva elementy se nesmějí vyskytovat vzájemně,
- oba elementy jsou povinné v předepsaném pořadí,
- oba elementy jsou povinné, ovšem mohou se vyskytovat v libovolném pořadí.

Ke každému elementu je možné přiřadit definice jeho atributů. Každý atribut je specifikován svým jménem, datovým typem nebo případně výčtem možných hodnot (enumerací). Navíc je nutné určit, zda je tento atribut povinný, nebo nepovinný.

V případě vynechání nepovinného atributu přiřazuje klientská aplikace takovému atributu implicitní hodnotu. Atributy mohou mít i pevně definované hodnoty.

V DTD je možné vymezit i booleovské atributy. Jejich jediná možná hodnota odpovídá názvu atributu. Takové atributy lze potom v HTML zapisovat v jejich zkrácené formě. Stačí napsat pouze samotný název bez hodnoty. Pokud se booleovský atribut v elementu vyskytuje, je mu přiřazena hodnota „pravda“ (a naopak).

1.3.2.2. Struktura dokumentu

Každý validní HTML dokument se musí skládat alespoň ze tří částí. Jedná se o řádek s informací o verzi jazyka, deklarativní hlavičku a tělo dokumentu. Informace o verzi jazyka zahrnuje i odkaz na DTD, vůči kterému lze dokument validovat. Hlavička slouží k vyjádření dodatečných informací o dokumentu. Jedná se v podstatě o metadata. Typickým příkladem informací, které se mohou vyskytovat v hlavičce, je jméno autora dokumentu, klíčová slova pro využití indexovacími službami, ekvivalenty parametrů HTTP hlaviček pro využití webovými servery a podobně.

V těle dokumentu se pak nalézá samotný obsah. Ten může být různým způsobem reprezentován jednotlivými prohlížeči, ať už jde o grafické, textové nebo jiné prohlížeče. V těle dokumentu se v zásadě mohou vyskytovat dva různé druhy elementů. Jde o elementy blokové *(block-level)*, nebo řádkové *(inline)*. Tyto dva modely vystihují dva základní typy chování elementů v rámci těla dokumentu. V zásadě platí, že blokové elementy mohou obsahovat buď další blokové elementy, nebo řádkové elementy, zatímco řádkové elementy mohou obsahovat pouze text, nebo jiné řádkové elementy. Odlišnosti najdeme i v jejich formátování. Zatímco výskyt blokového elementu sám o sobě determinuje nový řádek, řádkové elementy tuto vlastnost nemají (pokud není jejich klasické chování přepsáno ve stylové definici).

Z důvodů podpory světových jazyků je důležité, aby HTML dokumenty uměly zobrazovat text i zprava doleva. V tomto případě dochází k odlišnému chování blokových a řádkových elementů.

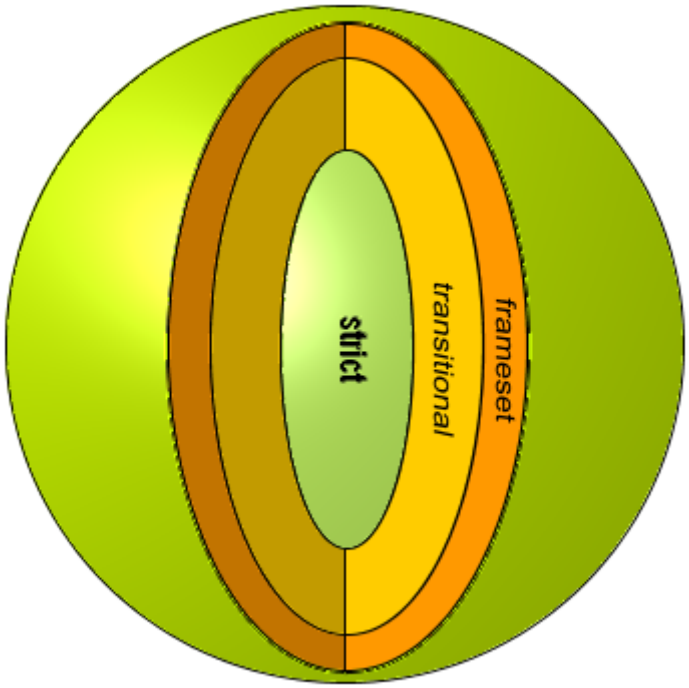
Omezení obsahu blokových a řádkových elementů je plně deklarováno v DTD. Některé elementy obsahují navíc další restrikce. DTD tak v některých případech neumožňuje blokovým elementům obsahovat některé další blokové či řádkové elementy a řádkovým elementům některé jiné řádkové. Takováto pravidla je možné plně postihnout v DTD pomocí hierarchických množin elementů, které lze v definicích libovolně skládat a kombinovat.

Klasickým představitelem blokového elementu je element `<div>` a řádkového `<span>`. Tyto elementy tvoří v kombinaci se stylovými definicemi obecný mechanizmus přidávání struktur do HTML dokumentů. Jejich jedinou vlastností je to, že dodržují blokový, nebo řádkový model. Všechny další (hlavně prezentační) vlastnosti mohou být volně definovány ve stylových definicích. Ty je pak možné jednoduše měnit (například vytvořit více verzí), což přináší značnou flexibilitu v prezentační rovině dokumentu.

1.3.2.3. Strict, Transitional, Frameset

Specifikace HTML 4.01 definuje jazyk ve třech mutacích a pro každou z nich existuje samostatná DTD definice. Jejich vzájemný vztah přibližuje obrázek 1.2 – „Vztah mutací jazyka HTML“. Tento vztah neplatí úplně do důsledku, ale s trochou tolerance můžeme

říci, že se v podstatě jedná o podmnožiny. Nejmenší množina se nazývá *strict*. Vymezuje HTML tak, jak jej jeho tvůrci skutečně zamýšleli. Oproti dřívějším verzím jazyka odpadla celá řada konstruktů, hlavně v důsledku oddělení většiny prezentačních informací, které se z praktických důvodů staly součástí stylových definic. Tato verze zahrnuje také nejvíce omezení a klade největší nároky na autory, kteří jsou zvyklí používat některé konstrukty z předešlých verzí jazyka. Tyto konstrukty pak většinou obsahuje množina s názvem *transitional*. Ta umožňuje autorům plynulý přechod k HTML 4.01, protože z důvodu kompatibility některé konstrukty zachovává. Ty jsou ovšem často označeny jako *depricated*. Takovým elementům budeme dále říkat potlačené. Potlačené elementy není doporučeno v nových dokumentech používat. Poslední množina zvaná *frameset* obsahuje navíc podporu rámů, které umožňují zobrazovat více HTML segmentů v rámci jednoho dokumentu.



Obrázek 1.2. Vztah mutací jazyka HTML

Hlavní důraz je kladen právě na dodržování syntaxe v rámci množiny *strict*. Ta má zaručit správné oddělení prezentační vrstvy od zbytku dokumentu. To přináší řadu výhod. Například možnost jednoduché změny vzhledu dokumentu bez zásahu do jeho zdroje. Děje se tak pouze v důsledku změny stylové definice.

1.3.2.4. Datové typy

V DTD definicích jazyka HTML je možné používat pouze primitivní datové typy vycházející ze SGML. Následující seznam ukazuje nejdůležitější z nich.

Vybrané datové typy jazyka SGML používané v DTD definicích jazyka HTML

PCDATA
: *Parsed Character Data* – V DTD definici jazyka HTML se takto označují elementy, které obsahují čistý text (čili posloupnost znaků). Pokud jsou navíc deklarovány nějaké subelementy, mohou být míchány s tímto textem.

CDATA
: *Character Data* – V HTML se takto označují hlavně atributy obsahující dále neomezený text (posloupnost znaků). Případné značky obsažené v atributech s tímto datovým typem nejsou nijak dále interpretovány. Ovšem znakové entity jsou nahrazeny příslušnými znaky. Konce řádek a tabulátory jsou nahrazeny mezerou.

MNTOKEN
: Atribut typu MNTOKEN může obsahovat libovolná písmena, číslice nebo speciální znaky. Jde například o pomlčku, podtržení, dvojtečku a nebo tečku.

NAME
: Hodnota omezená tímto datovým typem musí splňovat omezení kladená na datový typ MNTOKEN, navíc musí začínat malým, nebo velkým písmenem.

ID
: Musí splňovat stejná omezení jako datový typ NAME, ovšem navíc lze na takovýto identifikátor odkazovat pomocí dalšího datového typu IDREF. Proto musí platit, že hodnota atributu s datovým typem ID musí být jedinečná v rámci celého dokumentu.

NUMBER
: Sekvence číslic, která obsahuje minimálně jednu číslici.

Relativně expresivní je datový typ definovaný výčtem možných hodnot (enumerace). Hodnota atributu je tedy omezena na některou z deklarovaných hodnot.

1.3.3. Další omezení nad rámec DTD

Ne všechna omezení je možné zachytit v DTD. Některá z nich jsou pouze slovně vyjádřena v textu specifikace. V zásadě existují dva stupně volnosti dodržování těchto omezení. Specifická klíčová slova vyjadřují, zda je popisované omezení závazné a nebo zda jde pouze o doporučení, jehož nedodržení může sice působit komplikace, ale neznamená nutně nesoulad se standardem. Taková klíčová slova a jejich významy vymezuje [RFC2119].

1.3.3.1. Datové typy

Důležitou oblastí, kde se rozchází text specifikace a definice v rámci DTD, jsou datové typy viz. sekce 1.3.2.4 – „Datové typy“. Je to dáno hlavně vyjadřovací silou jazyka DTD. Ten umožňuje přiřazovat elementům a atributům pouze jednoduché datové typy ze SGML a enumerace. Další omezení kladená na jazyk HTML je možné vyjádřit pouze

slovně v textu specifikace. Tím je v podstatě znemožněna automatická kontrola dodržování těchto omezení.

Například atributy typu „univerzální identifikátory zdrojů“ (URI – *uniform resource identifier*), které slouží k identifikaci různých externích zdrojů v rámci různých druhů odkazů v dokumentu, mají přesnou syntaxi viz. [RFC2396]. DTD ovšem jejich formát dále nevymezuje. Umožňuje zadat libovolný text splňující omezení kladená na datový typ CDATA.

V mutacích *trasitional* a *frameset* se občas vyskytuje atribut popisující barvu některého z elementů. Mutace *strict* již počítá s tím, že takováto informace je čistě prezentační, a jako taková bude tedy zanesena ve stylové definici. Zatímco DTD popisuje tyto atributy jako CDATA, specifikace jasně definuje povolené hodnoty. Barva může být vyjádřena třemi hexadecimálními čísly (00 až FF, čili celkem 6 znaků) určujícími podíl červené, zelené a modré (RGB). Těmto číslům předchází znak „#“. Dále je možné jako hodnotu atributu zapsat jméno barvy. Pod jménem je rozeznáváno 16 základních barev (Black, Silver, Aqua ...).

Dalším datovým typem používaným v HTML jsou délky. DTD je opět specifikuje pouze jako CDATA. Délka nějakého elementu zobrazovaného v rámci HTML dokumentu může být vyjádřena pomocí procent a nebo v bodech. Délka musí být tedy zapsána jako kladné celé číslo. Pokud je toto číslo navíc zakončeno znakem „%“, pak se jedná o délku v procentech. Některé délky je možné specifikovat i relativně. Pokud máme více elementů, které spolu soupeří o nějaký volný prostor na stránce, může být jejich délka zapsána relativně k tomuto volnému prostoru. Takovou informaci zapisujeme kladným celým číslem následovaným znakem „*“. Pokud zapíšeme pouze znak „*“, automaticky se chápe jako „1*“. Číslo před hvězdičkou udává kolik dílů volného prostoru zaplní element s tímto atributem. Tak jako v předešlých případech i zde platí, že případné chyby v zápisu takových atributů není možné automaticky detekovat pomocí validace oproti DTD.

Pokud chceme do HTML dokumentu vnořit nějaký externí objekt nebo na něj odkazujeme, je potřeba specifikovat typ jeho obsahu. Toho je docíleno atributem, který obsahuje tzv. MIME *(Multipurpose Internet Mail Extention)* typ. Takový typ by měl odpovídat některému z typů registrovaných a spravovaných organizací IANA. Jméno typu se skládá z jména příslušného média odděleného lomítkem od jména formátu, ve kterém je toto médium uloženo.

Ukázka některých často používaných typů obsahu

text/html
: HTML dokument.

image/png
: Obrázek ve formátu PNG.

text/css
: Kaskádové styly.

Dalším druhem informace, která se vyskytuje v atributech elementů jazyka HTML, je informace o kódování znaků. Ta je vyjádřena jako jméno příslušné kódové tabulky. Tato jména jsou spravována v registru organizace IANA.

Podobně fungují i atributy popisující přirozený jazyk použitý pro obsah určitého elementu nebo jazyk, ve kterém je psán celý dokument. Takový atribut potom obsahuje příslušný kód přirozeného jazyka. Doporučení [RFC3066] se zabývá kódy jazyků, které je možné používat v rámci HTML dokumentů. V zásadě se kód jazyka skládá z povinného primárního kódu a libovolného množství nepovinných sekundárních kódů oddělených vzájemně pomlčkou. Primární kódy jsou rezervované pro zkratky jazyků podle [ISO639]. Každý sekundární kód je chápán jako kód země podle [ISO3166].

Zajímavým prvkem HTML jsou přístupové klávesy, které umožňují přímou aktivaci některých prvků dokumentu pomocí klávesnice. Takto lze označit elementy, jako jsou různé formulářové prvky a nebo odkazy. Hodnotou takového atributu je jeden libovolný znak, který odpovídá příslušné aktivační klávese. Další možností je specifikovat tuto hodnotu jako znakovou entitu. DTD definuje pro tyto atributy datový typ CDATA.

Pro některé elementy HTML je možné specifikovat datum a čas. Jedná se o elementy `<ins>` a `<del>`, které umožňují zachovávat přehled o změnách při revizích dokumentů. Každá zachycená změna tak může obsahovat informaci o datu a čase, kdy k této změně došlo. Takováto časová informace musí být zapsána v následujícím formátu:

YYYY-MM-DDThh:mm:ssTZD

YYYY
: Rok zapsaný čtyřmi číslicemi.

MM
: Dvouciferné pořadové číslo měsíce v roce, čili 01 odpovídá měsíci lednu a podobně.

DD
: Dvouciferné pořadové číslo dne v měsíci.

hh
: Dvouciferné označení hodiny.

mm
: Dvouciferné označení minuty.

ss
: Dvouciferné označení sekundy.

TZD
: Určení časové zóny. Například velké „Z“ označuje UTC *Coordinated Universal Time*. „+hh:mm“ specifikuje o kolik je lokální čas před časem UTC a „-hh:mm“ naopak o kolik je za ním.

Tento formát je (kromě jiných) popsán v [ISO8601]. Tato struktura časové značky je definována pouze v textu specifikace. V DTD se jedná o typ CDATA.

Pro odkazy v dokumentech lze specifikovat o jaký typ odkazu se jedná. Tuto informaci potom mohou využívat prohlížeče nebo indexující algoritmy vyhledávacích služeb. Takovéto informace lze vyjádřit v atributech jako seznam jednotlivých typů oddělených mezerami. Kompletní seznam možných typů je vyjádřen v textu specifikace.

Některé vybrané typy odkazů definovaných v HTML dokumentech

Alternate
: Alternativní verze dokumentu. Může jít například o verzi přeloženou do jiného jazyka nebo verzi v jiném formátu než je HTML.

Stylesheet
: Odkazuje na stylovou definici. V kombinaci s typem Alternate ukazuje alterativní styl dokumentu, který může být vybrán uživatelem.

Start
: Ukazuje na první dokument v kolekci dokumentů.

Next
: Ukazuje na další dokument z lineární sekvence dokumentů.

Prev
: Ukazuje na předchozí dokument z lineární sekvence dokumentů.

Contents
: Ukazuje na dokument sloužící jako obsah.

Autoři dokumentů mohou definovat i další typy odkazů nad rámec těch typů, které jsou vyjmenované ve specifikaci. Hodnotu takových atributů nelze tedy omezit výčtem.

Odkazy v HTML dokumentech mohou obsahovat atribut definující média, pro která je tento odkaz určen. Tímto způsobem se dá například přiřadit dokumentu různý styl podle toho, je-li například určen k tisku a nebo se bude zobrazovat na obrazovce. HTML explicitně definuje seznam klíčových slov, které lze použít pro definici vhodnosti pro dané médium. V atributech je potom vhodnost vyjádřena jako seznam těchto klíčových slov oddělených čárkou. Povolené názvy médií jsou následující.

screen
: Neodstránkovaná počítačová obrazovka.

tty
: Médium s neproporcionálními znaky. Jde například o terminál.

tv
: Médium podobné televizi. Má nízké rozlišení a limitovaný posun stránek.

projection
: Projektory.

handheld
: Kapesní počítače.

print
: Odstránkované dokumenty určené k tisku.

braille
: Dokument je čten pomocí zařízení, které interaguje s uživatelem pomocí Braillova písma.

aural
: Dokument je čten zařízením syntetizujícím lidskou řeč.

all
: Dokument je vhodný pro všechny typy médií.

DTD definuje pro takovéto atributy datový typ CDATA. I když je množina povolených označení médií pevně dána, specifikace nezakazuje do atributu zapisovat i jiné hodnoty, které nejsou definovány. Obsahuje-li atribut nějaký název média, který není definován ve specifikaci, má být prohlížečem ignorován. Pokud název média obsahuje nějaké jiné znaky než písmena, čísla nebo pomlčku, je zbytek názvu ignorován. Takže například „handheld with resolution 320x320“ má být interpretováno jako médium „handheld“.

V dokumentech, kde používáme rámy, je možné pro odkazy specifikovat, v jakém rámu se má odkazovaný dokument zobrazit. Jména takových rámů mohou obsahovat pouze písmena. To se ovšem netýká několika rezervovaných jmen, které mají specifický význam.

_blank
: Odkaz se otevře v novém okně.

_self
: Odkaz se otevře ve stejném rámu.

_parent
: Odkaz se otevře v přímém předkovi aktuálního rámu.

_top
: Zruší se rámová struktura a odkaz se otevře v aktuálním okně.

1.3.3.2. Hlavička dokumentu

V hlavičce HTML dokumentů je zajímavý element `<title>`. Tento element je povinný. Specifikace jasně vyzývá autory dokumentů, aby kvůli dostupnosti používali titulky, které budou popisovat co nejpřesněji obsah těchto dokumentů. V tomto případě se jedná o příklad omezení, které nebude možné kontrolovat automatizovaně ani s využitím expresivnějších validačních jazyků než je DTD. Specifikace totiž v tomto případě neposkytuje exaktnější definici vhodnosti příslušného titulku.

1.3.3.3. Tělo dokumentu

Elementy obsažené v těle dokumentu mají určitou společnou skupinu atributů. Mezi tyto atributy patří i atribut `class`. Ten definuje společnou třídu několika elementů v dokumentu. Pokud je takových tříd více, jsou jejich jména oddělena mezerou. Takovým třídám může být později přiřazen specifický způsob prezentace v rámci stylové definice. DTD definuje tento atribut jako CDATA.

1.3.3.4. Text

V části věnované textu se specifikace zabývá pravidly pro zobrazování a zápis mezer. Autoři dokumentů by neměli spoléhat na prohlížeče, že zobrazí mezeru, která je zapsána bezprostředně po počáteční značce, nebo bezprostředně před značkou koncovou. Proto je důležité, aby autoři pro vnořené řádkové elementy raději psali mezery mezi slovy vně těchto elementů (nikoli uvnitř). Takovéto mezery by nemusely být zobrazeny. Tento požadavek nelze zachytit v DTD.

Autoři dokumentů jsou odrazováni od toho, aby využívali prázdných elementů `<p>`. Elementy `<p>` slouží k definici odstavce textu. V tomto případě se nejedná o striktně vyžadované omezení, jde pouze o doporučení.

Element `<q>` označuje řádkové citace v textu dokumentu. Vizuální prohlížeče by měly označit takový text příslušným citačním znaménkem podle toho, jaký přirozený jazyk je aktuálně používán. Z tohoto důvodu je potřeba, aby se autoři zdrželi označování textu vlastními citačními znaménky, které by se tak objevovaly duplicitně. Toto omezení není formalizováno v DTD.

Elementy `<ins>` a `<del>` slouží k uchování historie změn provedených v HTML dokumentech. Zajímavostí těchto elementů je, že nejsou jednoznačně vymezeny. Podle místa jejich použití se chovají jako blokové, nebo jako řádkové elementy. To je ovšem v rozporu s DTD. Podle něj mohou tyto elementy vždy obsahovat jak blokové, tak i řádkové prvky. Následující příklad tedy bezchybně projde validací vůči DTD, ale specifikace takové použití zakazuje.

Příklad 1.4. Ukázka komentářů

```
<p>
  text ... <ins> <div> ...obsah blokového elementu... </div> </ins> ... ▶
text
</p>
```

Oblast specifikace zabývající se textem obsahuje ještě mnoho dalších omezení. Ta jsou však převážně adresována autorům HTML prohlížečů a zabývají se způsobem zobrazení textu v HTML dokumentech. Tato omezení jsou nad rámec této práce.

1.3.3.5. Seznamy

Pro seznamy je typické, že obsahují řadu potlačených atributů. Ty není doporučeno nadále používat. Týkají se prezentace seznamů a jejich místo je tedy ve stylové definici.

Specifikace pro seznamy neuvádí žádná dodatečná omezení, která by nebyla podchycena v DTD.

1.3.3.6. Tabulky

Tabulky obecně umožňují řadit data do řádků a sloupců. Specifikace nám radí, abychom nepoužívali tabulky k definici rozvržení stránek. K tomuto účelu by měla sloužit stylová definice. Tabulka by skutečně měla sloužit pouze pro reprezentaci tabulkových dat.

HTML specifikace doporučuje autorům, aby vytvářeli vystihující shrnutí obsahu tabulek. Tato shrnutí by měla pomoci uživatelům nevizuálních prohlížečů k lepšímu pochopení obsahu tabulek. To platí zvláště pro tabulky bez specifikovaného titulku.

Zajímavé je, že autoři dokumentů mohou správným zápisem tabulek v HTML pomoci zefektivnit jejich zobrazování pomocí prohlížečů. Například vyžadované umístění zápisu patičky tabulky před vlastním obsahem těla tabulky a nebo pevné vymezení počtu řádek a sloupců umožňuje prohlížečům vygenerovat tabulku najednou a pak ji jen postupně doplňovat daty. Prohlížeče takto tedy nemusí čekat s vykreslováním až na kompletní načtení všech dat týkajících se příslušné tabulky. Mohou tabulku vykreslovat postupně a umožnit uživatelům co nejrychlejší přístup k datům. Tento princip se nazývá inkrementální vykreslování.

Inkrementální vykreslování tabulek je obtížné i v případě, že definujeme velikost sloupců proporcionálně vzhledem k šířce tabulky a tabulka nemá pevně danou velikost. Stejně problematická je i situace, pokud autor nespecifikuje šířku sloupce vůbec. Řešení tohoto problému je však možné i na straně prohlížečů (hlavně grafických). Ty mohou při inkrementálním vykreslování jednotlivé sloupce postupně roztahovat podle potřeby.

Specifikace exaktně definuje, jakým způsobem mají prohlížeče stanovovat počet sloupců v tabulkách. V zásadě existují dva přístupy. Pokud autor HTML dokumentu definuje explicitně sloupce a skupiny sloupců, je možné určit tento počet z nich. V opačném případě je celkový počet sloupců určen jejich maximálním počtem v řádce. Podstatné je, že je považováno za chybu, pokud si v případě definování sloupců obě zmíněná čísla neodpovídají. Tento požadavek není zachycen v DTD.

Atribut označující šířku jednotlivých sloupců a skupin sloupců tabulek může (oproti klasickým atributům označujícím šířku) obsahovat i zápis „0*“. Takový zápis by za normálních okolností znamenal vyčlenění prostoru nulové šířky. Prohlížeče ovšem zobrazí takovéto sloupce s minimální šířkou tak, aby se do nich vešel celý jejich obsah. V této chvíli je nutné podotknout, že přiřazení takovéto hodnoty tomuto atributu může znemožnit prohlížečům zobrazovat tabulku inkrementálně. Je to dáno tím, že minimální šířku pro udržení obsahu je možné zjistit až po přečtení všech položek ve sloupci.

Přestože atribut určující šířku tabulek není potlačen, specifikace nedoporučuje jeho používání. Šířka i ostatní prezentační informace mají své místo ve stylové definici.

Za chybu se považuje i definování překrývajících se buněk. Toho je možné dosáhnout například s použitím atributů `rowspan` a `colspan`. Ten určuje roztažení určité buňky tabulky přes více buněk. Mohou se tak navzájem křížit buňky roztažené přes více sloupců s těmi roztaženými přes více řádek.

1.3.3.7. Odkazy

Odkazy jsou v podstatě spojením jednoho webového zdroje s jiným. Pomocí odkazů spolu můžeme asociovat různé zdroje různých typů.

Pomocí mechanizmů souvisejících s odkazy je možné označit nějaké místo v dokumentu, ke kterému se chceme v rámci stejného dokumentu odkazovat. Takové místo označujeme jako zakotvení. Jméno takového místa musí být podle specifikace unikátní v rámci celého dokumentu, a to nezávisle na malých a velkých písmenech ve jméně. Tento požadavek není zachycen v DTD. Zde se jedná o datový typ CDATA. Také platí, že autoři by měli chápat odkazy na taková místa v dokumentech jako citlivá na velikost písmen, aby se vyvarovali případných komplikací při použití některých prohlížečů.

Pokud autoři dokumentů odkazují na cizojazyčný zdroj, je pro lepší zacházení s takovým zdrojem ze strany prohlížečů možné specifikovat přirozený jazyk, ve kterém je tento zdroj napsán. Tento atribut je možné použít pouze v případě, že je nastaven i jiný atribut odkazu, který specifikuje to, na jaký zdroj vlastně odkaz ukazuje. Tento podmíněný výskyt atributu je za hranicemi vyjadřovací schopnosti DTD.

Zakotvení může být v dokumentech definováno na libovolném elementu pomocí atributu `id`. Na některých elementech, jako je třeba `<a>`, lze zakotvení definovat i pomocí atributu `name`. Pro oba tyto atributy platí, že sdílejí stejný jmenný prostor a musí být tedy společně unikátní v rámci dokumentu.

Elementy, které mohou obsahovat atribut `name`, mohou obsahovat zároveň i atribut `id`, ovšem jejich hodnota musí být stejná. Tento požadavek nelze postihnout v DTD.

1.3.3.8. Vkládané objekty a obrázkové mapy

Tato část se bude zabývat elementy umožňujícími vkládat prvky multimediálního rázu do HTML dokumentů. V dřívějších verzích HTML se používal element `<img>` pro vkládání obrázků a element `<applet>` pro vkládání java-appletů (specifických aplikací běžících na klientské straně). Element `<applet>` byl nyní potlačen ve prospěch obecnějšího elementu `<object>`, kterým je možné vkládat kromě obou zmíněných médií i celou řadu jiných objektů.

V zásadě se dá říci, že nároky kladené specifikací na vkládané objekty je možné automaticky validovat pomocí DTD. Výjimkou je snad jen atribut `archive`. Ten vyžaduje seznam URI identifikátorů, zatímco jeho datový typ v DTD je omezen pouze na CDATA.

V HTML dokumentech je možné definovat mapy oblastí (typicky oblasti obrázku) a přiřazovat jim specifické akce pro případ, že je příslušný region aktivován uživatelem. HTML rozeznává dva druhy takovýchto obrázkových map. První je mapa na straně klienta. Při použití této mapy jsou souřadnice aktivované uživatelem předány prohlížeči, který na základě nich vybere příslušný odkaz. V případě map na straně serveru jsou souřadnice posílány serverovému agentovi, jehož umístění je specifikováno jedním z atributů takové mapy. Způsob interpretace souřadnic je pak dán implementací tohoto agenta.

Pro obrázkovou mapu definuje specifikace zajímavé omezení, které je třeba zmínit. Pokud definujeme souřadnice některého z regionů mapy, formát těchto souřadnic se liší podle toho, jaký tvar oblasti specifikujeme v atributu `shape`. Například pro kružnici zadáváme souřadnice středu a poloměr. Polygon naopak vyžaduje sérii dvojic souřadnic,

kde poslední dvojice odpovídá souřadnicím dvojice první. Tím je dosaženo řádného uzavření polygonu. Ovšem odpovědnost za uzavření polygonu leží na bedrech prohlížečů. Takováto omezení nelze specifikovat v DTD.

Pro obrázkové mapy i pro vkládané objekty jsou důležité alternativní popisy. Ty zlepšují dostupnost pro uživatele nevizuálních prohlížečů. Ovšem obecně platí, že pokud není možné obsah nějakého objektu zobrazit, může alternativní popis výrazně zlepšit orientaci v dokumentu. Proto jsou tyto popisy většinou povinné a specifikace navíc ukládá autorům následující doporučení pro jejich používání. V některých případech je lepší vyplnit jako alternativní popisek pouze prázdný řetězec. Například v případech, kdy obrázek slouží pouze jako určitý dekorativní prvek bez dalšího významu. Alternativní popis by tedy vždy měl přinášet nějakou smysluplnou informaci. Popis typu „bla bla“ nebo „bezvýznamný obrázek“ by mohl být matoucí pro uživatele nevizuálních prohlížečů.

1.3.3.9. Styly

HTML dokumenty mohou obsahovat stylové informace přímo v sobě. Lze je zapsat v některém ze stylových jazyků, jako je například kaskádový styl (CSS). Mohou také odkazovat na styly externí. Druhý zmíněný způsob přináší jistě větší flexibilitu, neboť umožňuje dynamicky měnit styl dokumentu, aniž bychom museli do dokumentu zasahovat. Navíc je možné sdílet určitý styl mezi více dokumenty.

V prvním případě ovšem specifikace vyžaduje, aby autoři určili použitý stylový jazyk. Pokud se v dokumentu vyskytuje alespoň jeden atribut `style`, který umožňuje zapsat vnitřní stylovou definici pro daný element, je nutné nastavit použitý stylový jazyk v deklarativní hlavičce, a nebo je nutné, aby tuto informaci poskytoval webový server v HTTP hlavičce. Dokument, který tuto informaci neposkytuje, je chybný. To platí i přesto, že prohlížeče by měly implicitně předpokládat stylový jazyk CSS (kaskádový styl).

1.3.3.10. Vizuální formátování

Většina elementů a atributů, která byla dříve užívána k vizuálnímu formátování obsahu HTML dokumentů, byla nyní potlačena ve prospěch stylů. I přesto bude dobré zmínit některá omezení, která na ně klade specifikace a která není možné automaticky validovat s využitím DTD. Takové je například i omezení týkající se atributu určujícího velikost písma.

Tuto velikost je možné specifikovat relativně, nebo absolutně. V druhém případě označuje symbol „+“ a „-“ to, zda se má velikost písma zvětšit či zmenšit o hodnotu za tímto symbolem. Výsledné velikosti ovšem vždy náleží do intervalu 1 až 7. Relativní změna velikosti se váže k základní velikosti písma, která je definována pro celý dokument. Pokud tato hodnota není autorem explicitně nastavena, automaticky se předpokládá hodnota 3.

1.3.3.11. Formuláře a ovládací prvky

Formuláře přinášejí HTML dokumentům možnost interakce s uživatelem. Formuláře mohou kromě jiného obsahovat i ovládací prvky. Těmi jsou různá textová pole, výběry,

tlačítka a další. Takto je možné implementovat určitý druh uživatelského rozhraní přímo v HTML dokumentech.

Základním elementem pro definici ovládacích prvků je element `<input>`. Ten může definovat více typů různých ovládacích prvků. Specifikace explicitně vyžaduje (na rozdíl od DTD), aby byla v případě prvků typu *„checkbox“* a *„radio“* povinně specifikována jejich hodnota. Pro ostatní typy ovládacích prvků je tato hodnota dobrovolná. Existují ještě další pravidla, kde výskyt některých atributů elementu `<input>` závisí na zvoleném typu ovládacího prvku. Ovšem specifikace nezakazuje přímo autorům tyto atributy v takovém případě použít, ale naopak nařizuje HTML prohlížečům tyto atributy ignorovat.

Tlačítka lze kromě elementu `<input>`, definovat i pomocí elementu `<button>`. Ten přináší další možnosti, a to hlavně možnost vkládat do takového tlačítka další HTML obsah. Například obrázky s textem a podobně. Ne všechny elementy uvnitř takového tlačítka jsou povolené. Jejich vymezení je definováno v DTD. Ovšem jedno pravidlo v DTD podchyceno není. Pokud tlačítko obsahuje nějaký obrázek, tak tento obrázek nesmí být asociován s žádnou obrázkovou mapou.

Element `<select>` specifikuje výběr z určitého seznamu možností. Každá možnost může být definována jako implicitně vybraná. Jelikož není jasně specifikováno, jakým způsobem se prohlížeče mají chovat, pokud není žádný prvek implicitně vybrán, specifikace doporučuje (nepřikazuje) autorům označit alespoň jednu možnost jako výchozí. Chybou ovšem je, pokud je v seznamech bez možnosti mnohonásobného výběru označeno více prvků jako vybrané.

Zajímavou problematikou je navigace mezi jednotlivými ovládacími prvky ve formulářích. Jednou z možností, jak lze aktivovat některý prvek, je klávesa „tabulátor“. Pořadí aktivace jednotlivých prvků při opětovném stisku této klávesy závisí na atributu `tabindex`. Specifikace jasně deklaruje, že hodnota tohoto atributu musí být z intervalu 0 až 32767. Toto omezení není zachyceno v DTD.

1.3.3.12. Skriptovací jazyky v HTML

Skriptování umožňuje dynamický přístup k objektovému modelu HTML dokumentů a v některých případech i dynamické generování obsahu dokumentů. Skripty mohou být volány při načtení dokumentu a nebo při nějaké specifické události, jako je například stisknutí tlačítka. HTML je samo o sobě nezávislé na použitém skriptovacím jazyce. Jeho podpora je dána konkrétním prohlížečem. Pokud je v dokumentu použito skriptování v závislosti na události, je nutné, aby autoři specifikovali implicitní skriptovací jazyk pro daný dokument v jeho hlavičce. Tuto informaci může poskytovat i webový server v HTTP hlavičce. Pokud ovšem tato informace není dostupná v některé z těchto forem, jedná se o chybný dokument.

1.4. XHTML 1.0

Specifikace XHTML 1.0 je v podstatě přepracováním HTML 4.01 tak, aby splňovala omezení kladená na jazyk XML *(Extensible Markup Language)*. XHTML je tedy aplikací XML. Množina elementů a atributů je plně poděděna z HTML 4.01. Kromě omezení, která vznikla z důvodu dodržování přísnější syntaxe jazyka XML, definuje specifikace

ještě několik doporučených postupů, které mají zajistit zpětnou kompatibilitu XHTML dokumentů s HTML prohlížeči.

Přechod na XML zajišťuje jazyku HTML nemalé výhody. Autoři dokumentů i prohlížečů mohou k editaci i zpracování XHTML dokumentů využívat velkou škálu nástrojů, aplikací a knihoven, které XML podporují. Těch je vzhledem k oblíbenosti tohoto jazyka v dnešní době skutečně velké množství. K nim se řadí i validační nástroje, které budou předmětem další kapitoly. Důvodem, proč je XML tak populární, je pravděpodobně to, jakým způsobem zachovává původní vyjadřovací schopnosti SGML, a přitom snižuje jeho přílišnou komplexitu. Striktní syntaxe XML nedovoluje některé možnosti zápisu elementů a atributů, které byly možné v případě HTML odvozeného přímo ze SGML. Tato striktnost výrazně ulehčuje práci aplikacím zpracovávajícím XHTML, jako jsou například prohlížeče. Tato skutečnost je významná například pro mobilní zařízení s malým výpočetním výkonem.

Dokumenty striktně splňují specifikaci XHTML, pokud splňují všechna mandatorní omezení uvedená ve specifikaci. Takový dokument musí odpovídat některé DTD definici v rámci XHTML 1.0. Kořenový element musí mít definován odpovídající jmenný prostor a odpovídajícím způsobem musí být deklarován i typ dokumentu s odkazem na použité DTD. Jmenné prostory přinášejí zajímavou možnost kombinovat XHTML s jinými XML jazyky, jako je například SVG, používané pro vyjádření vektorové grafiky, nebo MathML, používané pro vyjádření matematických vzorců.

1.4.1. Rozdíly mezi HTML a XHTML

Novinkou je, že XHTML dokumenty musí být řádně formovány (well-formed). To například znamená, že všechny elementy musí mít ukončovací značku, nebo musí být zapsány speciálním způsobem, aby bylo zřejmé, že ukončovací značka následovat nebude (zpětné lomítko na konci značky). Správné musí být i vnořování elementů. Ty se stejně jako v HTML nesmějí navzájem překrývat.

Vzhledem k tomu, že jména elementů a atributů v XML rozlišují velká a malá písmena, jsou všechna jména atributů a elementů v XHTML předepsána jako malá. V HTML na jejich velikosti nezáleželo.

V HTML se hodnoty atributů mohly uzavírat relativně volně viz. 1.3.1 – „Základní konstrukty jazyka SGML používané v HTML 4.01“. V XHTML je jediným povoleným způsobem uzavření ve dvojitých uvozovkách. Hodnoty booleovských atributů navíc nemohou být zkracovány. Každý vyznačený atribut musí obsahovat nějakou hodnotu, byť by to byl prázdný řetězec.

DTD popisující omezení jazyka XML (XML DTD) není v některých případech tak expresivní jako DTD pro HTML. Nelze v něm vyjádřit některá omezení. Tato omezení přesto platí, ale není možné je v případě XHTML dokumentů automaticky validovat. Příkladem je element `<a>`. Ten nesmí obsahovat sám sebe na libovolné úrovni vnoření. Takových elementů je celá řada. Specifikace XHTML obsahuje jejich úplný seznam viz. http://www.w3.org/TR/2002/REC-xhtml1-20020801/#prohibitions.

Na rozdíl od XHTML nerozlišuje HTML velikost písmen v možných hodnotách enumerací, proto jsou v XHTML všechny takové možnosti definovány v malých písmenech.

1.4.2. Zpětná kompatibilita XHTML

Specifikace XHTML obsahuje sekci s doporučeními, jakým způsobem lze postupovat, aby XHTML dokumenty byly zpětně kompatibilní s nástroji pracujícími s HTML. Taková doporučení nemají vliv na validitu dokumentů, a proto zde zmíníme pouze několik ilustrativních příkladů. Pro prázdné elementy je doporučeno přidat mezeru na konec jejich jména, autoři by se měli vyvarovat odřádkování v hodnotách atributů a podobně.

1.5. ISO-HTML

ISO-HTML je de facto mezinárodní standard, na jehož tvorbě se podílela Mezinárodní standardizační organizace (ISO) a Mezinárodní elektrotechnická komise (IEC). Blízce spolupracovalo i W3C. Ostatně ISO-HTML vychází z doporučení W3C pro HTML 4.01 viz. [HTML4]. ISO-HTML respektuje všechna omezení v tomto doporučení a přidává navíc některá vlastní viz. [ISO15445]. Cílem standardizace ISO-HTML je definovat určité jádro jazyka HTML, které bude obsahovat jen obecně podporované funkce, aby dokumenty splňující tyto standardy byly zobrazitelné na co největším okruhu prohlížečů a platforem.

V zásadě platí, že mnoho elementů je převzato bez jakýchkoli změn z doporučení W3C z mutace *strict* a některé elementy jsou upraveny o další restrikce. Mnoho atributů je vypuštěno, protože popisují spíše prezentaci a ne význam elementů. ISO-HTML se zbavuje všech potlačených prvků, všech vlastností, jejichž role je pouze kosmetická, a také prvků, které jsou nestabilní nebo nevyzrálé. Také byly vypuštěny některé volitelné konstrukty, jako je například vynechávání některých značek. Syntaxe jazyka se tak stala striktnější i na té nejnižší úrovni. ISO-HTML také není svázáno požadavky na zpětnou kompatibilitu, jako jsou doporučení W3C.

1.5.1. Omezení ISO-HTML

V této části se budeme zabývat pouze některými zajímavými omezeními kladenými na ISO-HTML. Hlavně se soustředíme na omezení, která nelze zapsat v DTD. Dále nás budou zajímat výrazné odlišnosti od doporučení W3C.

V zásadě platí, že soubor elementů je v ISO-HTML stejný jako v HTML 4.01 *strict*. Některé elementy byly ovšem upraveny a mnoho atributů bylo vypuštěno. Oproti doporučení W3C se v ISO-HTML mění definice blokových a řádkových elementů. Blokové elementy jsou v zásadě převzaty, ovšem nemohou obsahovat jednotlivá záhlaví (např. `<h1>`) a element `<address>`.

Namísto řádkových elementů *(inline)* se zde zavádí elementy označované jako *text*. Tento obsahový model v zásadě odpovídá řádkovému modelu z HTML 4.01, ovšem neobsahuje elementy reprezentující ovládací prvky.

Kromě strukturálních změn, jako je zákaz výskytu některých elementů v určitém kontextu, zahrnuje specifikace ISO-HTML oproti doporučením W3C ještě některé další změny.

Atribut `coord` elementu `<a>` má být vynechán, pokud je atribut `shape` nastaven na výchozí hodnotu *(default)*.

Od autorů je dále striktně vyžadováno, aby v elementu `<blockquote>` nepoužívali explicitní citační znaménka (např. uvozovky). Zobrazení těchto znamének je plně v kompetenci prohlížeče.

Pro element `<button>` existuje zajímavý požadavek. Pokud je jeho typ nastaven na „submit“, pak jsou atributy `name` a `value` povinné.

Deklarativní hlavička dokumentů nemůže obsahovat element `<script>`, neboť standard neshledává skriptovací technologie za dostatečně stabilní funkci HTML. Nestabilní technologie standard raději nepřipouští.

ISO-HTML se podrobně věnuje záhlavím. Specifikuje některá omezení tak, aby byla záhlaví reprezentovaná elementy `<h1>` až `<h6>` lépe použitelná pro popis struktury dokumentu. ISO-HTML chápe záhlaví spíše jako způsob označení sekcí v rámci dokumentu. Striktně vyžaduje, aby byla vždy dodržena následnost těchto sekcí. Po každé sekci označené jako `<h1>` musí tedy následovat jedině další `<h1>`, nebo `<h2>`, a nikoli například `<h4>`. Doporučení W3C se k této problematice vyjadřuje pouze vágně s tím, že někteří lidé považují přeskakování sekcí za špatné. Ovšem na rozdíl od ISO-HTML nestanovuje žádná normativní omezení této praxe.

Požadavek na nepřeskakování sekcí není vyjádřen v DTD pro ISO-HTML, protože kontrola tohoto omezení by vyžadovala definici dodatečných pomocných elementů, které by nebyly součástí jazyka. Proto existuje speciální přípravná DTD definice, která obsahuje některé dodatečné kontrolní elementy. Ty jsou ale při publikaci z jazyka vypuštěny, aby byl výsledný dokument korektním ISO-HTML. Takový mezikrok je součástí doporučené publikační strategie ISO-HTML, ovšem nebyl by nutný při validaci s využitím moderních validačních jazyků (viz. další kapitoly).

Mezinárodní standard požaduje, aby obrázky obsahovaly atribut `ismap` pouze v případě, jsou-li uzavřeny v elementu `<a>`, který má specifikován atribut `href`. Dále platí, že atributy `usemap` a `ismap` se nesmí objevit současně.

Mnoho změn zaznamenaly ovládací prvky. Například tlačítko může být specifikováno pouze elementem `<button>`, a nikoli pomocí elementu `<input>`. Navíc pro různé typy ovládacích prvků jsou dovoleny a vyžadovány různé atributy. Kompletní seznam je obsažen v [ISO15445].

Pro styly vyžaduje mezinárodní standard, aby byl implicitní stylový jazyk deklarován v hlavičce dokumentu. Doporučení W3C přitom povoluje i možnost zasílat tuto informaci v HTTP hlavičce webovým serverem.

1.6. WAI WCAG 1.0

WAI *(Web Accessibility Initiative)* je iniciativa při W3C, která se snaží koordinovaně s organizacemi po celém světě pomoci webu a HTML dokumentům, aby byly dostupnější pro všechny, tedy i pro osoby s různými postiženími a podobně. Kromě jiných aktivit vydává i směrnice týkající se dokumentů HTML. Ty budou právě předmětem našeho zájmu.

Směrnice pro dostupnost webového obsahu WCAG *(Web Content Accessibility Guidelines)* viz. [WCAG1] jsou určeny primárně autorům HTML dokumentů a autorům publikačních nástrojů. Dodržování těchto standardů nepřináší jenom větší dostupnost pro uživatele s postiženími, ale obecně zlepšuje dostupnost pro všechny uživatele bez

ohledu na platformu a nebo prohlížeč. Další oblastí, kde mají směrnice pomoci, je lepší vyhledávání dokumentů.

Směrnice WCAG řeší v podstatě dva základní okruhy problémů. První okruh se zabývá dostupností dokumentů v různých prostředích a s různými omezeními na straně uživatele. Jde například o fyzická, senzorická a jiná postižení nebo pracovní a technologické bariéry. Obecně se dá říci, že v této oblasti pomáhá mimo jiné: separování prezentační a strukturální vrstvy, využívání textu (ten je totiž dostupný v podstatě v jakémkoli prostředí), poskytování informací ve vizuálních i v nevizuálních kanálech a vytváření stránek nezávisle na hardwaru (malá obrazovka, bez myši apod.). Těmito problémy se zabývají směrnice 1 až 11.

Směrnice 12, 13 a 14 se zbývají druhým okruhem problémů. Jejich cílem je učinit dokumenty srozumitelné a lehce navigovatelné. Autoři by měli vybavit dokumenty navigačními nástroji a informacemi pro lepší orientaci v dokumentech.

Jednotlivá doporučení v rámci směrnice jsou opatřena prioritami. Existují tři základní prioritní stupně. Splnění směrnice s prioritou 1 je naprostou nutností. Pokud není takové doporučení splněno, bude dokument nedostupný pro některé skupiny uživatelů. Priorita 2 říká, že nesplnění tohoto doporučení nevede sice k nedostupnosti dokumentu, ale značně ztěžuje jeho dosažení určitým skupinám uživatelů. Doporučení s prioritou 3 vede obecně ke zlepšení dostupnosti dokumentů.

Dodržené priority determinují stupeň souladu dokumentu se směrnicemi. Dokumenty s označením „Triple-A“ splňují všechny priority, dokumenty „Double-A“ první dvě priority a dokumenty označené „A“ jen prioritu 1.

WAI neposkytuje automatickou možnost validace svých směrnic. Díky pokročilým validačním jazykům jsou některé části těchto směrnic předmětem doplňkové validace v rámci této práce.

1.6.1. Směrnice 1 – Alternativy pro vizuální i auditivní obsah

Vizuální prvky by měly být popsány textovými ekvivalenty tak, aby význam jejich popisu plnil stejnou funkci jako vizuální prvek samotný. Text může být přečten hlasovými syntetizery, může být vypsán v Braillově písmu a nebo může být formátován v libovolných velikostech. Text je tedy velice dostupným médiem, a proto je kladen důraz na textové ekvivalenty. Z předchozího plynou následující omezení pro HTML dokumenty.

Každý netextový element musí obsahovat příslušný textový ekvivalent. To se týká obrázků, obrázkových map, grafické reprezentace textu (včetně symbolů), animací, programových objektů, ASCII „obrázků“, rámů, skriptů, obrázkových seznamových odrážek, grafických tlačítek a audio či video souborů *(priorita 1)*.

Navíc obrázkové mapy by měly obsahovat nadbytečné textové odkazy pro každý definovaný region *(priorita 3)*. Multimediální video a audio sekvence by měly být opatřeny synchronizovaným textovým popisem důležitých informací v nich obsažených *(priorita 1)*.

1.6.2. Směrnice 2 – Nespoléhejte se pouze na barvy

Pokud je jediným nositelem nějaké informace barva, může se lehce stát, že nebude dostupná pro uživatele, kteří nerozeznávají některé barvy nebo tyto barvy neumí interpretovat jejich hardware *(priorita 1)*. Problematické může také být, pokud se barva pozadí málo kontrastně odlišuje od barvy textu *(priorita 2 pro obrázky a 3 pro text)*.

1.6.3. Směrnice 3 – Správné používání značek a stylů

Nesprávné užívání značek zahrnuje jejich zneužívání k prezentaci (využití tabulky k rozvržení stránky) a zneužívání prezentačních značek k vyjádření struktury (tabulky vytvořené elementem `<pre>`).

Pokud existuje vhodné značkování, autoři by mu vždy měli dát přednost před vyjádřením této informace pomocí obrázku. To platí například pro text, jehož prezentace může být vždy vhodně modifikována pomocí stylového jazyka.

Prezentace a rozvržení stránek by měla být definována ve stylech, a nikoli v dokumentu pomocí značek. Délky v atributech elementů a ve stylu by měly být pokud možno relativní (nikoli absolutní). Relativní délky se dají lépe přizpůsobit různým rozlišením zobrazovacích zařízení.

Pro popis struktury dokumentů je dobré využívat záhlaví v různých úrovních, správně značkovat seznamy a správně značkovat citace. Citací by se nemělo zneužívat k formátovacím účelům, jako je například odsazení *(všechna omezení v této směrnici mají prioritu 2)*.

1.6.4. Směrnice 4 – Přirozený jazyk

Kvůli hlasovým syntentizerům je dobré označit hlavní jazyk dokumentu v jeho hlavičce *(priorita 3)* a také vyznačit přechody do jiných přirozených jazyků v rámci textu a textových ekvivalentů *(priorita 1)*. Stejně tak je dobré vyznačit vysvětlení zkratek v místě, kde se poprvé objeví v textu *(priorita 3)*. Tento přístup není vhodný jen pro uživatele s postiženími, ale může být využit i vyhledávacími službami.

1.6.5. Směrnice 5 – Správně se zobrazující tabulky

Tabulky by měly skutečně být používány pouze k označení tabulkových dat, a nikoli pro rozvržení stránek *(priorita 2)*. V datových tabulkách je nutné identifikovat záhlaví sloupců a řádek značkami `<td>` a `<th>` *(priorita 1)*. V tabulkách, které mají více logických stupňů a složitější vazby mezi daty, je třeba tyto vazby zachytit příslušnými značkami *(priorita 1)*. Tabulky by měly obsahovat shrnutí a zkratky nadpisů v záhlaví *(priorita 3)*.

1.6.6. Směrnice 6 – Interpretace nepodporovaných technologií

Ačkoli jsou autoři dokumentů podporováni v používání nových technologií, které mnohdy odstraňují problémy s těmi staršími, je nutné, aby udržovali určitou zpětnou kompatibilitu se staršími prohlížeči, které tyto technologie nepodporují, a aby zachovali dostupnost pro uživatele, kteří se rozhodli mít tyto technologie vypnuty.

Dokumenty by měly být čitelné i bez přiloženého stylu *(priorita 1)*. Ekvivalenty dynamického obsahu by měly být řádně aktualizovány, pokud se jejich obsah změní *(priorita 1)*. Dokumenty by měly být použitelné i bez podpory skriptování, appletů a podobných programových elementů *(priorita 1)*. Ošetření událostí v dokumentech (jako je aktivace myší apod.) by mělo být nezávislé na použitém typu vstupního zařízení *(priorita 2)*. Je třeba zajistit dostupnost dynamického obsahu, nebo zajistit jeho alternativní prezentaci *(priorita 3)*. Příkladem je využitím elementu `<noframes>` na konci každé definice rámů.

1.6.7. Směrnice 7 – Obsah citlivý na čas

Ne všichni uživatelé mohou vnímat rychle se měnící informace v dokumentech. Takovou informací je například pohybující se text. Proto by vždy měly být dostupné mechanizmy pro ovládání takového pohybu.

Autoři stránek by měli zamezit blikání a nebo třesu obrazu *(priorita 1)*. Také by se měli vyvarovat poblikávání obsahu *(priorita 2)*. Pokud se v dokumentu vyskytuje nějaký pohyblivý obsah, měl by mít uživatel zároveň možnost tento pohyb zastavit. Stejně tak by měl mít možnost zastavit automatické obnovování a přesměrování dokumentu. Přesměrování by se navíc nemělo definovat značkami v dokumentu, ale spíše pomocí konfigurace webového serveru *(priorita 2)*.

1.6.8. Směrnice 8 – Dostupnost vnořených uživatelských rozhraní

Pokud mají objekty vnořené do dokumentu nějaké uživatelské rozhraní, pak by toto rozhraní mělo být uživatelům dostupné *(pokud je jejich funkcionalita důležitá a nedostupná na jiném místě, jde o prioritu 1, jinak se jedná o prioritu 2)*.

1.6.9. Směrnice 9 – Dostupnost nezávislá na vstupním zařízení

Autoři by měli spíše užívat obrázkové mapy na klientské straně. Ty mohou být různými druhy prohlížečů zpracovány adekvátním způsobem *(priorita 1)*. Každý prvek s vlastním rozhraním by měl být ovladatelný metodami nezávislými na vstupním zařízení. Stejně tak události v dokumentech by měly být ošetřeny nezávisle na vstupním zařízení *(priorita 2)*. Autoři by měli zajistit logické pořadí aktivování ovládacích prvků pro lepší navigaci pomocí tabelátoru. Důležité odkazy by měly mít definovanou klávesovou zkratku *(priorita 3)*.

1.6.10. Směrnice 10 – Přechodná řešení

Kvůli zpětné kompatibilitě a lepší dostupnosti je někdy nutné používat některá přechodná řešení. Tak jak se budou jednotlivé prohlížeče vyvíjet, je naděje, že tato přechodná omezení přestanou být významná.

Autoři by neměli využívat „vyskakující“ okna a neměli by měnit stávající obrazovku bez upozornění uživatele. Implicitní popisky ovládacích prvků by měly být správně umístěny. Lze je umístit těsně před prvkem na stejném řádku nebo bezprostředně o řádek výše *(priorita 2)*. Pro prohlížeče, které neumí zobrazovat bloky textu vedle sebe

na šířku, je potřeba doplnit tabulky příslušnými alternativami s formátováním pod sebou. Některé starší prohlížeče ignorují některé prázdné ovládací prvky. Je proto potřeba tyto prvky vyplnit nějakým „místo zabírajícím“ znakem (například elementy `<input>` a `<textarea>`). Jiné starší prohlížeče zase neumí řádně zobrazovat přiléhající odkazy. Je proto dobré mezi ně vložit nějaký tisknutelný znak ohraničený mezerou *(priorita 3)*.

1.6.11. Směrnice 11 – Používání technologií a směrnic W3C

Autoři by měli užívat technologie W3C, pokud jsou dostupné a vhodné pro daný účel. Měli by používat poslední verze, pokud jsou široce podporovány. Autoři by se měli zdržet užívání potlačených prvků těchto technologií *(priorita 2)*. Užitečné je používat možnosti popisu obsahu pomocí metadat, což může pomoci uživatelům, aby dostali skutečně ty dokumenty, které chtějí *(priorita 3)*.

Pokud není z nějakého důvodu možné vytvořit dostupný dokument s použitím technologií a směrnic W3C, je nutné vytvořit alespoň odkaz na dokument, který tyto předpoklady splňuje. Tento dokument musí obsahovat ekvivalentní informace a musí být aktualizován stejně často jako dokument původní *(priorita 1)*.

1.6.12. Směrnice 12 – Kontextové a orientační informace

Každý rám je nutné opatřit titulkem pro lepší navigaci *(priorita 1)*. Pokud to není jasné z titulku, je dobré opatřit rámy delším textem, který bude popisovat účel rámu a vztah k ostatním rámům *(priorita 2)*.

Velké bloky informací je žádoucí, tam kde to má smysl, seskupovat do menších celků. Stejně tak je rozumné přiřazovat ovládacím prvkům explicitní popisky značené pomocí elementu `<label>` *(priorita 2)*.

1.6.13. Směrnice 13 – Zřetelný navigační mechanizmus

Tato problematika zahrnuje navigační lišty, mapy stránek a podobné prvky, které mohou být užitečné pro všechny uživatele.

Autoři by měli jasně identifikovat odkazy tak, aby jejich text poskytoval jasnou informaci o obsahu, na který odkaz ukazuje. Uživatelé by měli k dokumentům připojovat i metadata, která popisují obsah těchto dokumentů. Součástí stránek by měla být navigační komponenta popisující rozvržení informací. Takovou komponentou je například mapa stránek nebo obsah. Navigační komponenty na různých místech stránek by měly mít vždy konzistentní ovládací rozhraní *(priorita 3)*. Jendou z navigačních komponent by měl být i navigační panel *(priorita 3)*.

Pokud stránky obsahují vyhledávání, mělo by být různě nastavitelné s různými stupni komplexity. Pokud existuje více dokumentů, které spolu mají určité logické vazby, měl by tento fakt být popsán s použitím atributů elementu `<link>` *(priorita 3)*.

1.6.14. Směrnice 14 – Pochopitelné a srozumitelné dokumenty

Směrnice doporučuje autorům používat co nejsrozumitelnější a nejjasnější vyjadřování odpovídající obsahu dokumentu *(priorita 1)*. Kde to může pomoci pochopení, je dobré přidat do dokumentu obrázky a zvuky. Je také dobré vytvořit konzistentní stylovou definici pro všechny související dokumenty *(priorita 3)*.

1.6.15. Shrnutí

Z výše uvedeného je zřejmé, že mnoho omezení definovaných v [WCAG1] je definováno natolik vágně, že je nelze už z principu popsat i v těch nejexpresivnější validačních jazycích. I přesto je však možné některé části směrnic formalizovat. To umožňuje automatizovanou kontrolu těchto omezení, což může vést k lepšímu dodržování některých doporučení v praxi. Dokument, který úspěšně projde takovouto validací, ovšem nebude možné považovat za dokument, který je v souladu se směrnicemi. Automatizovaná kontrola může sice usnadnit práci a zrychlit proces vývoje vyhovujících dokumentů, ale lidský faktor zůstane neoddělitelnou součástí validace.

Kapitola 2

Analýza validačních jazyků přesahujících schopnosti DTD a posouzení vhodnosti jejich použití pro doplňkovou validaci

2.1. Úvod

V dnešní době existují desítky různých validačních jazyků, které validují XML dokumenty. Liší se od sebe stupněm dospělosti, rozšířeností, čitelností, podporou (jsou například zaštiťovány některou organizací s významným slovem v oblasti IT), oblíbeností v uživatelské komunitě a samozřejmě vyjadřovací schopností a přístupem k popisu omezení obecně.

V této práci není možné porovnávat vlastnosti a vhodnost všech validačních jazyků, kterých jsou desítky. Je nutné učinit určitý výběr, a ten teprve podrobit důslednějšímu zkoumání. Do tohoto výběru byly zvoleny jazyky: XML DTD, XML Schema, Relax NG, Schematron a DSD. V následujících několika odstavcích bude vysvětleno, jaké zajímavé vlastnosti jednotlivých jazyků vedly k jejich vybrání.

2.1.1. XML DTD

Je zřejmé, že XML DTD nebude tím vyvoleným jazykem, který by poskytl lepší validaci HTML. Naopak jeho vyjadřovací schopnost je ještě o něco málo horší než u klasického SGML DTD, neboť se jedná o jeho podmnožinu viz. sekce 1.4 – „XHTML 1.0“. Tento jazyk bude v analýze sloužit jako určitý srovnávací element. Je zřejmé, že pokud je naším cílem rozšířit validační schopnosti nad rámec DTD a budeme chtít popsat celý jazyk HTML v některém alternativním validačním jazyce, musí tento jazyk obsahovat minimálně schopnosti jazyka DTD. Díky přítomnosti DTD v analýze tak bude možné jednoduše porovnat přidanou hodnotu toho kterého jazyka. DTD je navíc vysoce lidsky čitelné a velice rozšířené. I přes rychlý nástup validačních jazyků postavených na XML je XML DTD pravděpodobně stále nejrozšířenějším validačním jazykem. Velkou nevýhodou tohoto jazyka je fakt, že sám není aplikací XML, což znemožňuje využití některých nástrojů při jeho zpracování. Ztěžuje to také možnost kombinování s jinými jazyky a znamená to, že XML DTD není jazykem sebepopisujícím. Navíc jeho vyjadřovací síla je v porovnání s konkurenty slabá. V tomto dokumentu budeme vycházet z verze jazyka 1.0. Další informace viz. [XMLDTD].

2.1.2. XML Schema

XML Schema je validační jazyk z dílny W3C postavený na XML. Právě zaštítění ze strany W3C slibuje tomuto jazyku velkou životaschopnost. Tento jazyk velice rychle penetruje do praxe a je oblíben v uživatelské komunitě. Existuje mnoho nástrojů, které ho podporují. Záměrem W3C je učinit z XML Schema de facto nástupce XML DTD, oproti kterému je XML Schema mnohem expresivnější a přináší některé nové mechanizmy. Ty zahrnují například promyšlený a komplexní systém datových typů nebo některé mechanizmy známé z objektového programování, jako je například dědičnost. Ta pomáhá modularizovat jednotlivé definice a umožňuje jejich znovupoužitelnost. XML Schema se postupně stává v podstatě průmyslovým standardem mezi validačními jazyky, které validují XML. V tomto dokumentu budeme vycházet z verze jazyka 1.0. Další informace viz. [XMLSCH-ST] a [XMLSCH-DT].

2.1.3. Relax NG

Tento jazyk vzniká pod patronací mezinárodního konsorcia OASIS. V nedávné době prošel i mezinárodní standardizací v rámci ISO. Relax NG vychází ze dvou validačních jazyků. Jde o TREX, vyvinutý Jamesem Clarkem, a RELAX, navržený Makoto Muratou. RELAX NG vyniká hlavně jednoduchostí použití, dobrou rozšiřitelností (například je možné použít různé knihovny datových typů) a hlavně výbornou čitelností. RELAX NG totiž může být zapsán jak ve své XML verzi, tak i v podobě speciálně navržené pro snadnou čitelnost. Dá se tedy říci, že Relax NG si zachovává jednoduchost a čitelnost DTD a zároveň přináší výhody XML. Elegantně je řešena i modularita definic. Relax NG je pro svou jednoduchost a elegantnost kladně přijímán uživatelskou komunitou, a proto již v dnešní době existuje celá řada nástrojů, které se svým okruhem a funkcionalitou rychle vyrovnávají těm, které fungují pro XML Schema. V tomto dokumentu budeme vycházet z verze jazyka 1.0. Další informace viz. [RNG].

2.1.4. Schematron

Schematron používá naprosto odlišný přístup k validaci dokumentů než všechny výše zmíněné jazyky. Zatímco tyto jazyky se orientují na definování kompletní gramatiky popisovaného jazyka, Schematron definuje pouze sérii jednotlivých testů pro jednotlivá omezení. Nemusíme tak definovat všechny elementy a vazby mezi nimi, ale stačí se z praktického hlediska soustředit na konkrétní omezení, která chceme kontrolovat. Schematron není vyvíjen žádnou velkou organizací (prochází ovšem standardizačním procesem) a za jeho zrodem stojí jediný muž, Rick Jelliffe. Ten geniálním způsobem skloubil možnosti již existujících technologií (XSLT – *eXtensible Stylesheet Language Transformations* a XPath), aby vytvořil zcela unikátní validační jazyk. Definice tohoto jazyka je velice jednoduchá. Obsahuje jen několik elementů. Pokud tedy uživatel již ovládá XPath, není velkým problémem naučit se i Schematron. Schematron je v podstatě syntézou již existujících, široce používaných a standardních technologií, což mu zajišťuje vysokou životaschopnost. Schematron má výbornou vyjadřovací sílu. Lze v něm popsat některé závislosti, které by byly v jiných jazycích nepopsatelné. Naopak vyjádření všech strukturálních vazeb v určitém typu XML dokumentů je pomocí

Schematronu poměrně neohrabané. Čitelnost Schematronu je velmi dobrá, pokud čtenář ovládá XPath. V opačném případě se může lehce orientovat podle slovních vyjádření, která by měla popisovat každý jednotlivý test i soubor testů. Schematron v zásadě nepotřebuje žádné speciální nástroje. Validace pomocí Schematronu je sérií dvou standardních XSLT transformací, které se běžně užívají pro XML dokumenty. V této práci budeme vycházet z verze jazyka 1.5. Schematron je v zásadě nezávislý na verzi jazyka XPath. Ačkoli již existuje verze 2.0, není ještě široce podporována, a proto budeme v této práci vycházet z XPath 1.0. Další informace viz. [SCHTR].

2.1.5. DSD

Posledním vybraným jazykem je jazyk DSD *(Document Structure Description)*. Ten je zaštiťován telekomunikačním gigantem AT&T, který na něm pracuje společně s výzkumnou katedrou počítačové techniky na Aarhuské universitě. DSD je zajímavý hlavně tím, že stojí kdesi mezi jazyky popisujícími kompletní strukturu a těmi co používají jednotlivá testová omezení. V DSD je totiž možné vyjádřit oboje. K vyjádření testových kritérií není ovšem použit XPath, jako je tomu v případě Schematronu, ale jsou využity spíše regulární výrazy a booleovská logika. To jistě zvyšuje vyjadřovací schopnost jazyka, ovšem neumožňuje takovou flexibilitu vyjádření jako XPath. DSD neobsahuje žádné předdefinované datové typy. Kontrola hodnot je plně zajišťována regulárními výrazy. Čitelnost se zdá být někde na úrovni XML Schema. Navíc čtenář musí ovládat regulární výrazy, aby pochopil některá omezení, která mohou být v XML Schema vyjádřena srozumitelnějším datovým typem (například URI). DSD se zdá být v uživatelské komunitě vcelku málo známé. Tomu odpovídají i podpora a nástroje. V tomto dokumentu budeme vycházet z verze jazyka 2.0. Další informace viz. [DSD].

2.2. Posouzení obecných schopností validačních jazyků

V této části se zaměříme na výčet schopností, které jsou obecně vlastní validačním jazykům a porovnáme podporu těchto vlastností v rámci námi vybraných kandidátů. Porovnání těchto základních vlastností je důležité, abychom si udělali přehled o celkových schopnostech a vyjadřovací síle vybraných jazyků. Cílem této části není pouze vybrat řešení, které popíše maximum omezení kladených na HTML dokumenty viz. první kapitola, ale je nutné vybrat i řešení, které bude nejsnadněji implementovatelné, dobře čitelné, pochopitelné a rozšiřitelné v případě, bude-li žádoucí implementovat některá další omezení. Například některé nově vznikající standardy W3C.

Dalším důležitým aspektem výběru vhodného jazyka je schopnost vyjádřit omezení modulárním způsobem. To bude velice důležité pro následnou formalizaci omezení jazyka HTML a XHTML a také pro implementaci prototypové aplikace, která by měla být schopná dynamicky validovat dokumenty jen oproti určité skupině omezení. Schopnost elegantně vyjádřit tato omezení v určitých vymezených modulech a tyto moduly následně jednoduše spojovat a rozdělovat bude tedy pro tuto práci klíčová.

Například mutace jazyka HTML/XHTML viz. sekce 1.3.2.3 – „Strict, Transitional, Frameset“ jsou definovány ve třech DTD definicích. Většina jednotlivých formalizačních zápisů se v těchto souborech vždy třikrát opakuje. Jednotlivé definice se liší jen málo. Přijetí takového konceptu by mělo za následek velice špatnou spravovatelnost definic.

Každá změna by se musela zapisovat až třikrát a definice by se musely neustále udržovat navzájem konzistentní. Proto je jistě lepším řešením vyjádřit v přídavných modulech pouze odlišnosti jednotlivých mutací.

2.3. Vlastnosti validačních jazyků

V následujícím textu je rozebrán určitý výčet podstatných vlastností validačních jazyků a jejich podpora mezi kandidáty pro doplňkovou validaci. Některé posuzované kategorie byly inspirovány prací [LEE-CHU].

2.3.1. Zabudované datové typy

Mnoho validačních jazyků podporuje určitou množinu vestavěných datových typů. Nejvíce datových typů poskytuje XML Schema. Jde o dobře vybranou množinu třiceti sedmi různých datových typů, které v zásadě odpovídají datovým typům používaným v programovacích jazycích. Další odvozené datové typy lze vytvářet pomocí mechanizmů podobných dědičnosti. XML DTD podporuje pouze deset základních datových typů, které vychází z datových typů jazyka XML. Relax NG má pouze dva základní datové typy, ovšem tato skupina je dále libovolně rozšiřitelná. Lze používat přídavné knihovny datových typů a existuje obecný mechanizmus, jakým se lze na tyto knihovny z jazyka odvolávat. Použití příslušné knihovny závisí na tom, zda-li je podporována naší konkrétní implementací validátoru. Typickým příkladem přídavné knihovny datových typů jsou datové typy jazyka XML Schema. Ty lze tedy používat i v Relax NG. Schematron je díky závislosti na XPath orientován hlavně na testování strukturálních závislostí v dokumentech. Určité chování podobné datovým typům je ovšem možné pomocí XPath simulovat. Například je možný test, zda-li některý z atributů daného elementu obsahuje nějaký konkrétní znak. Také lze například testovat, zda je určitý řetězec menší než deset znaků. Podobně fungují datové typy v DSD. Jeho tvůrci se domnívají, že jazyk je mnohem flexibilnější, pokud používá místo datových typů regulární výrazy. Těmi lze vyjádřit téměř libovolné omezení pro tvar určité hodnoty atributu. DSD tedy nemá žádné zabudované datové typy. Nevýhodou takového přístupu může být horší čitelnost a vyšší požadavky na znalosti uživatele.

Pokud se podíváme na datové typy jazyka HTML, je zřejmé, že by byly plně popsatelné v libovolném z výše uvedených validačních jazyků (kromě XML DTD a Schematronu). XPath 1.0 není tak bohatý, aby dokázal nahradit regulární výrazy. Zvládne pouze jednoduchá omezení v rámci základních typů používaných v XPath, což je řetězec, booleovská hodnota a číslo. Takové hodnoty mohou být dále zpracovávány pomocí jednoduchých funkcí. Silnější nástroje přináší až XPath 2.0.

Příklad 2.1. Příklad simulace datových typů v Schematronu

```
<assert test="(number(@isbn) > 999999999) and (number(@isbn) < ▶
1000000000)">
  ISBN je desetimístné číslo
</assert>
```

2.3.2. Atributy

2.3.2.1. Vylučovací výběr atributů

V praxi se občas hodí podmínit přítomnost jednoho atributu v elementu tím, že nebude přítomen i jiný atribut a naopak. Toto omezení není možné vystihnout v XML DTD ani v XML Schema. Naopak Relax NG využívá pro atributy stejný mechanizmus, jaký funguje pro podmíněný výskyt dvou elementů. Takovouto závislost není problém vyjádřit ani v případě Schematronu a DSD.

Příklad 2.2. Příklad vylučovacího výběru atributů. Osoba nemůže obsahovat věk a datum narození současně

```
<!-- příklad v Schematronu -->
<rule context="Osoba">
  <report test="@vek and @datumNarozeni">
    Osoba nemůže obsahovat atribut vek a datumNarozeni současně
  </report>
</rule>

<!-- příklad v DSD -->
<if><element name="Osoba"/>
  <declare>
    <attribute name="vek"/>
    <attribute name="datumNarozeni"/>
  </declare>
  <if><attribute name="vek"/>
    <require><not><attribute name="datumNarozeni"/></not></require>
  </if>
  <if><attribute name="datumNarozeni"/>
    <require><not><attribute name="vek"/></not></require>
  </if>
</if>

<!-- příklad v Relax NG -->
<element name="Osoba">
  <choice>
  <attribute name="vek">
   <text/>
  </attribute>
  <attribute name="datumNarozeni">
    <text/>
  </attribute>
  </choice>
</element>
```

Ukázka také hezky ilustruje rozdíl mezi jazyky, které popisují kompletní strukturu dokumentů, a jazyky, které využívají jednotlivých testů. DSD stojí mezi oběma koncepty. Organizace testů v DSD má blízko k deklarativní podobě definice, kterou vidíme na příkladu v Relax NG.

2.3.2.2. Podmíněný výskyt atributu na základě hodnoty jiného atributu

V některých případech je žádoucí, aby hodnota některého atributu byla podmíněna výskytem jiného atributu v elementu. Například specifikace HTML vyžaduje, aby byl v případě elementu `<input>` s typem *„radio“* nebo *„checkbox“* přítomen atribut `value`. Toto omezení je možné zapsat pouze ve Schematronu, DSD a Relax NG.

Příklad 2.3. Příklad popisu podmíněného výskytu atributu na základě hodnoty jiného atributu

```
<!-- příklad ve Schematronu (pouze pro typ radio) -->
<rule context="input">
  <report test="@type='radio' and not(@value)">
    Pokud atribut type obsahuje hodnotu 'radio' pak element input musí ▶
obsahovat atribut value
  </report>
</rule>

<!-- příklad v DSD (pouze pro typ radio) -->
<if>
 <and>
  <element name="input"/>
  <attribute name="type"><string value="radio"/></attribute>
 </and>
 <require>
  <attribute name="value"/>
 </declare>
</if>

<!-- příklad v Relax NG -->
<!-- ošetřeny jsou všechny povolené hodnoty atributu type -->
<element name="input">
  <choice>
   <group>
     <attribute name="type">
       <choice>
         <value>radio</value>
         <value>checkbox</value>
       </choice>
     </attribute>
     <attribute name="value"/>
```

```
    </group>
    <group>
      <attribute name="type">
        <choice>
          <value>text</value>
          <value>password</value>
          <value>submit</value>
          <value>reset</value>
          <value>hidden</value>
        </choice>
      </attribute>
      <optional>
        <attribute name="value"/>
      </optional>
    </group>
  </choice>
</element>
```

2.3.2.3. Podmíněný výskyt atributu na základě výskytu jiného atributu

V některých případech je potřeba, aby si přítomnost nějakého atributu vynutila i přítomnost jiného atributu. Například v HTML je pro element `<a>` vyžadována přítomnost atributu `href` v případě použití atributu `hreflang`. Takové omezení lze zapsat pouze v Schematronu, DSD a Relax NG.

Příklad 2.4. Příklad podmíněného výskytu atributu na základě výskytu jiného atributu

```
<!-- příklad v Schematronu -->
<rule context="a">
 <report test="(@hreflang) and not(@href)">
  Pokud element a obsahuje atribut hreflang, musí obsahovat i atribut href

 </report>
</rule>

<!-- příklad v Relax NG -->
<!-- nejsou definovány všechny atributy elementu a -->
<element name="a">
  <choice>
   <group>
     <attribute name="hreflang"/>
     <attribute name="href"/>
   </group>
   <group>
     <optional>
```

```
        <attribute name="href"/>
      </optional>
    </group>
  </choice>
</element>
```

2.3.3. Elementy

2.3.3.1. Různé druhy obsahu elementů

Elementy mohou být prázdné, mohou obsahovat jiné elementy, mohou obsahovat pouze text a nebo jiné elementy spolu s textem. Tuto funkcionalitu podporují všechny zkoumané jazyky, včetně Schematronu. Následující tabulka ukazuje vyjádření těchto základních typů obsahu v jednotlivých jazycích.

Tabulka 2.1. Typy obsahu

Typ	XML DTD 1.0	XML Schema 1.0	Relax NG 1.0	Schematron 1.5	DSD 2.0
prázdný element	`<!ELEMENT a EMPTY>`	`<content value="empty">`	`<empty/>`	`not(*)`	`<empty>`
text	`<!ELEMENT a (#PCDATA) >`	`<content value="textOnly">`	`<text/>`	`string-length(text()) > 0`	`<stringtype/>`
obsahuje jiné elementy	`<!ELEMENT a (x?|y*|z+) >`	`<content value="elementOnly">`	`<zeroOrMore> <element name="a"> </zeroOrMore>`	`count(element::*) = count(*)`	`<element>`
může obsahovat text i elementy	`<!ELEMENT a (#PCDATA|x)* >`	`<content value="mixed">`	`<zeroOrMore> <element name="a"> </zeroOrMore> <zeroOrMore> <text/> </zeroOrMore>`	toto platí implicitně	`<anyelement>`

2.3.3.2. Počet výskytu elementů

Některé validační jazyky umožňují přesně specifikovat počet výskytu některého z elementů. Například XML Schema umožňuje definovat minimální a maximální počet výskytů. Schematron je díky XPath vybaven podobnými možnostmi. Pomocí funkce `count()` lze přesně omezit počet výskytů. Stejně expresivní je i DSD, kde se užívá atributů `min` a `max`. XML DTD a Relax NG umožňují pouze vágní definici výskytu. Matematika DTD je značně omezená, umí počítat pouze do nuly, do jedné a do „moc“. Existují tedy tři možnosti: 1) žádný, nebo jeden výskyt, 2) žádný, nebo hodně výskytů, 3) jeden, nebo hodně výskytů. Podobně vágní je i Relax NG.

2.3.3.3. Podmíněný výskyt elementů

Jazyky postavené na definici kompletní struktury dokumentů, jako je Relax NG, DTD nebo XML Schema, mohou podmiňovat výskyt elementů na základě definice obsahu jiných elementů. Tímto způsobem lze jednoznačně určit vztah potomek-předek na základě výčtu všech povolených subelementů pro každý jednotlivý element. Specifikace

složitějších podmínek je obtížná. Ovšem DSD a Schematron disponují mnoha nástroji pro omezení výskytu elementů podle nejrůznějších pravidel a v závislosti na nejrůznějších podmínkách. Možností je nepřeberně. Například výskyt elementu v závislosti na hodnotě některého atributu, na počtu výskytů jiného elementu, na libovolných vlastnostech a nebo výskytu libovolně vzdáleného předka a podobně. DSD dosahuje takových omezení pomocí svého důmyslného aparátu booleovských podmínek a Schematron díky jazyku XPath.

2.3.4. Modularita

Tato část bude analyzovat možnosti validačních jazyků vzhledem k rozdělení definic omezení na drobné samostatné části. Bude také zkoumat možnosti jejich opětovného skládání do libovolných množin omezení dle aktuální validační potřeby.

2.3.4.1. Zahrnutí externích definic *(include)*

Tato funkcionalita nám umožňuje oddělit některé fragmenty definice do separátních souborů a následně je libovolně spojovat dohromady. Takový přístup zlepšuje čitelnost velmi dlouhých definic a umožňuje také jednoduše vytvářet více verzí omezení podle toho, které fragmenty do výsledné definice zahrneme. Zahrnutí externích definic je možné v XML DTD, XML Schema, Relax NG a DSD. Schematron sice přímo takovou funkcionalitu nepodporuje, ovšem díky jednoduchosti a otevřenosti návrhu Schematronu je ji možné lehce simulovat pomocí jím používaných technologií (XML, XSLT).

2.3.4.2. Dědění jednoduchých datových typů

Pokud máme modulární definici omezení, samotné poskládání jednotlivých fragmentů dohromady nám mnohokrát nestačí. Často je potřeba, aby nově zahrnutý fragment upravoval nějakým způsobem omezení již definovaná ve fragmentech ostatních a nebo z těchto omezení nějakým způsobem vycházel. Můžeme si tedy představit, že přidáním fragmentu potřebujeme spolu s ním zavést i nový datový typ. Ten se ovšem liší jen málo od některého datového typu, který již existuje v definici. Dědičnost tedy umožňuje rozšířit, a nebo omezit stávající definici datového typu.

Dobrou podporu pro dědičnost má XML Schema. Ovšem datové typy je možné pouze dále omezovat, a nikoli rozšiřovat. Relax NG umožňuje omezovat pouze zabudované datové typy, a nebo datové typy v přídavných knihovnách. Takto vzniklý typ není možné dále upravovat. Ovšem v Relax NG je možné i rozšiřovat enumerace o další povolené hodnoty. DSD umožňuje dále omezovat datové typy přidáním dalších booleovských podmínek. Podobně i Schematron může přidat další restrikce pomocí dodatečného testu. XML DTD se omezuje pouze na deset základních pevných datových typů a jejich expanzi ani restrikci neumožňuje. Kompletní přímou podporu rozšiřování datových typů neobsahuje žádný z uvedených jazyků (pouze nepřímo pomocí přepisu viz. dále).

2.3.4.3. Dědičnost a přepis definic elementů

Přidáním některého fragmentu do definice může nastat potřeba, aby tento fragment modifikoval některé stávající definice elementů. Pokud bychom například definovali omezení jazyka HTML a měli modul definic pro mutaci *strict* a k němu bychom zahrnuli ještě modul *transitional*, pak by bylo potřeba dodefinovat celou řadu elementů již definovaných ve *strict* tak, aby obsahovaly všechny prezentační atributy, které jsou v mutaci *transitional* navíc. Naopak v některých případech může vzniknout potřeba opačná. Přidáním některého fragmentu je potřeba omezit některé stávající definice elementů.

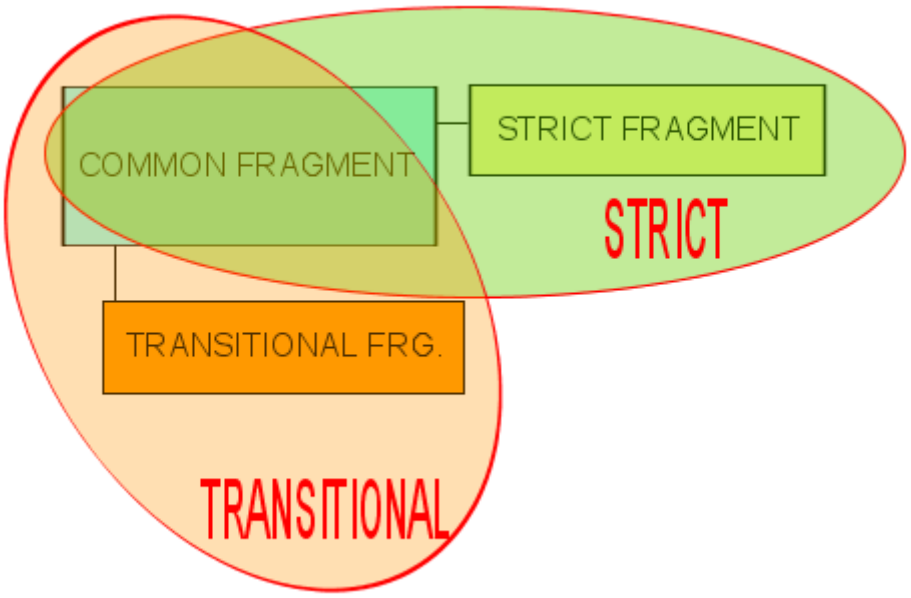
Obě předešlé možnosti jsou dostupné v XML Schema. XML Schema umožňuje díky elementu `<redefine>` zahrnout některý externí fragment a modifikovat libovolné definice v něm. Tento přístup má ovšem řadu omezení. Není například možné tyto definice kompletně přepsat, je možné je pouze rozšiřovat a nebo omezovat, což nemusí být vždy dostačující. Navíc tento mechanizmus umožňuje modifikovat pouze definice v zahrnovaném fragmentu. Zahrnovaný fragment nemůže ovlivnit definice ve fragmentu zahrnujícím.

Příklad 2.5.

```
<!-- Redefine v XML Schema, zákazník v main.xsd může obsahovat ještě ▶
element poznamka -->
fragment1.xsd:
<xs:complexType name="Osoba">
 <xs:sequence>
    <xs:element name="titul" minOccurs="0"/>
  <xs:element name="jmeno" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="adresat" type="Osoba"/>

main.xsd:
<xs:redefine schemaLocation="fragement1.xsd">
  <xs:complexType name="Osoba">
  <xs:complexContent>
    <xs:extension base="Osoba">
      <xs:sequence>
        <xs:element name="poznamka" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
  </xs:complexType>
</xs:redefine>

<xs:element name="zakaznik" type="Osoba"/>
```

Další možností, jak lze pracovat s modulární strukturou definic v XML Schema, jsou abstraktní typy. Jedná se o přístup podobný abstraktním třídám z OOP. Tímto způsobem lze definovat globální typy, které se dají použít až pokud jsou dodefinovány v některém ze zahrnutých fragmentů. Chceme-li tedy měnit definici některého elementu v závislosti na použitých fragmentech, je nejprve nutné definovat v nějakém společném fragmentu jednotlivé abstraktní typy. Ty mohou být posléze konkretizovány v přídavných fragmentech. Definice těchto elementů se pak může libovolně měnit podle toho, jakým způsobem konkretizujeme tyto typy v zahrnutých fragmentech, a které fragmenty zahrneme do definice.

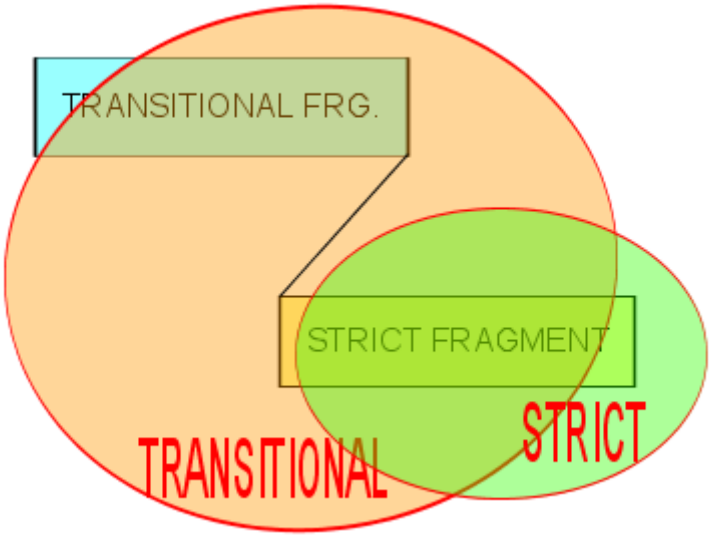
Pokud se vrátíme k původnímu příkladu s mutacemi *transitional* a *strict*, je zřejmé, že dva moduly definic nestačí. Budeme potřebovat jakousi hlavní definici, která bude deklarovat všechny společné elementy a připraví všechny abstraktní elementy, které se mohou v jednotlivých mutacích lišit viz. obrázek 2.1 – „Modularita definic HTML pro mutaci *strict* a *transitional* s fragmentem společných definic“. K této hlavní definici pak podle potřeby zahrneme fragment *transitional* nebo *strict*, které budou poskytovat konkrétní definice abstraktních elementů.



Obrázek 2.1. Modularita definic HTML pro mutaci *strict* a *transitional* s fragmentem společných definic

Oproti relativně těžkopádné modularitě definic v XML Schema přichází Relax NG s velice jednoduchým, ale vysoce expresivním konceptem. Expanze definic elementů pomocí zahrnutých fragmentů je řešena atributem `combine`. Jednotlivé fragmenty totiž mohou obsahovat více definic stejného jména. Atribut `combine` potom pouze určí, jakým způsobem se mají definice zkombinovat. Tento atribut určuje, zda se jedná o vylučovací výběr, sekvenci a nebo o elementy v libovolném pořadí. Restrikce již není tak přímočará. Dá se pro ni ovšem využít mechanizmus přepisu (viz. *overwriting* v OOP).

V místě zahrnutí externího fragmentu můžeme jednoduše přepsat libovolné definice z toho fragmentu na jejich restriktivnější verze. V příkladu s mutacemi jazyka HTML bychom tedy skutečně mohli mít pouze *strict* a *transitional* fragment. Pro validaci dokumentu psaného v mutaci *strict* bychom použili pouze *strict* fragment. Naopak v případě *transitional* bychom validovali oproti oběma fragmentům zahrnutým společně viz. obrázek 2.2 – „Modularita definic HTML pro mutace *strict* a *transitional* se samostatným fragmentem *strict*“. Definice tohoto zahrnutí by obsahovala případné restriktivní přepisy. Ovšem pouze pokud by mutace *transitional* kladla nějaká restriktivnější omezení oproti mutaci *strict*.



Obrázek 2.2. Modularita definic HTML pro mutace *strict* a *transitional* se samostatným fragmentem *strict*

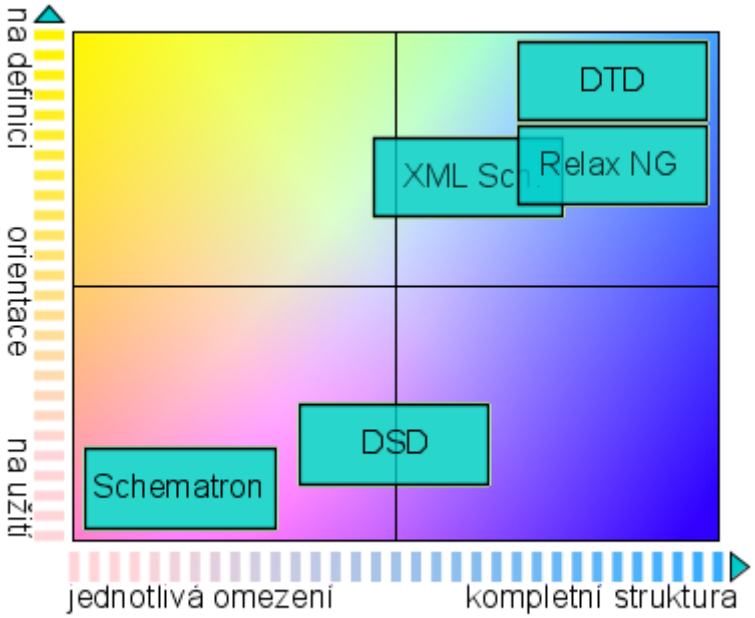
Schematron je dosti odlišný od předešlých dvou jazyků, a proto se zde omezíme pouze na to, že restriktivní modularita je v něm v zásadě možná přidáním dodatečných testů.

DSD také poskytuje jisté nástroje pro podporu dědičnosti a přepisu. Restriktivní dědičnost není z principu jazyka problém. Jednoduše se zahrnou separátní fragmenty, které budou přinášet dodatečná omezení pro jednotlivé elementy. Problém nastává v případě expanzivních změn. Expanzivní změny nejsou implicitně podporovány. Nepomůžeme si ani přepisem. Ten byl podporován elegantním způsobem ve verzi jedna pomocí atributu `renewId`. Odkazem na identifikátor příslušného elementu bylo možné přepsat celou definici, a dokonce přepsat i případné přepisy. Ovšem s restrukturalizací jazyka tento prvek ve verzi dvě vymizel.

Tabulka 2.2. Souhrnný přehled schopností validačních jazyků. Tabulka obsahuje i další vlastnosti, které nebyly zmíněny výše.

Vlastnost	XML DTD 1.0	XML Schema 1.0	Relax NG 1.0	Schematron 1.5	DSD 2.0
Obecné vlastnosti					
XML syntaxe	NE	ANO	ANO	ANO	ANO
Podpora jmenných prostorů	NE	ANO	ANO	ANO	ANO
Datové typy					
Zabudované datové typy	10	37	2-?	0	0
Rozšiřitelnost datových typů	NE	ANO	ANO	Částečná	ANO
Atributy					
Vylučovací výskyt atributů	NE	NE	ANO	ANO	ANO
Podmíněná hodnota atributu	NE	NE	ANO	ANO	ANO
Podmíněný výskyt atributu na základě hodnoty jiného atributu	NE	NE	ANO	ANO	ANO
Povinné a nepovinné atributy	ANO	ANO	ANO	ANO	ANO
Elementy					
Různé typy obsahu elementů	ANO	ANO	ANO	ANO	ANO
Pevné pořadí elementů	ANO	ANO	ANO	ANO	ANO
Neřazené sekvence elementů	Částečně (nutné definovat všechny možnosti)	ANO	ANO	ANO	ANO
Exklusivita (jeden vylučuje druhý)	ANO	ANO	ANO	ANO	ANO
Počet výskytu elementů	Částečně	ANO	Částečně	ANO	ANO
Podmíněný výskyt elementů	v zásadě NE	v zásadě NE	v zásadě NE	ANO	ANO
Modularita					
Zahrnutí externích definic	ANO	ANO	ANO	přímo NE	ANO

Vlastnost	XML DTD 1.0	XML Schema 1.0	Relax NG 1.0	Schematron 1.5	DSD 2.0
Dědění jednoduchých datových typů restrikcí	NE	ANO	NE	ANO	ANO
Dědění jednoduchých datových typů expanzí	NE	NE	NE	NE	NE
Dědění definic restrikcí	NE	ANO	ANO	ANO	ANO
Dědění definic expanzí	NE	ANO	NE	NE	NE
Přepis definic	NE	NE	ANO	NE	NE



Obrázek 2.3. Profily validačních jazyků (koncept je převzat z [LEE-CHU] a přizpůsoben této práci)

2.4. Čitelnost

S validačními jazyky úzce souvisí i čitelnost. Pokud negenerujeme a nezpracováváme definice omezení strojově, ale vytváříme je ručně, a tyto definice jsou značně rozsáhlé, což je přesně případ této práce, je čitelnost velice důležitým faktorem. S čitelností úzce souvisí také modularita, která byla rozebrána v předešlé části. Jazyk, který dovoluje rozdělit definice do malých úzce specifikovaných modulů a který nevytváří příliš velké náklady na jejich pozdější skloubení, obsahuje nezbytný základ k dobré čitelnosti rozsáhlých definic. Posouzení čitelnosti jednotlivých jazyků by mělo být považováno spíše za orientační, neboť čitelnost je vágní pojem, a není tedy možné jasně ohodnotit

nejčitelnější z jazyků. Každý jazyk může být vhodnější pro popis určitých typů omezení nebo struktur. Navíc stejná omezení mohou být zapsána mnoha různými způsoby, a proto by opravdu vypovídající srovnání muselo být značně rozsáhlé. Jednoduchý příklad zápisu stejných omezení v různých jazycích je uveden v příloze A – „*Definice záznamu o knize v jednotlivých validačních jazycích*“. Z této přílohy si lze udělat alespoň hrubý obrázek o tom, co formalizace omezení v jednotlivých jazycích obnáší a jak se liší rozsáhlost a čitelnost definic v případě popisu stejných omezení.

V zásadě se dá říci, že XML obecně čitelnosti příliš neprospívá. Čitelnost definic psaných v DTD a nebo v kompaktní formě Relax NG je velice dobrá. XML Schema je ve svých definicích dosti rozvláčné. Dá se říci, že zápis podobných struktur v Relax NG bude vycházet o něco lépe. Filozofie Schematronu je značně odlišná od ostatních jazyků. Schematron není vhodným jazykem pro popis kompletní struktury dokumentů. Spíše je vhodný pro vyjádření jednotlivých testů. Proto je zápis ve Schematronu v podstatě nejméně čitelný. Na druhou stranu existuje celá řada případů, které lze vyjádřit jednoduchým Schematronovým testem, zatímco v ostatních jazycích by byla jejich formalizace velice komplikovaná, pokud by byla vůbec možná. Někde mezi Schematronem a ostatními stojí DSD, které je určitým kompromisem a umožňuje popsat relativně čitelně oba zmíněné přístupy. Uspořádání jednotlivých testů v DSD má totiž blízko ke kompletním strukturálním definicím.

2.5. Kombinování více validačních jazyků

Pro doplňkovou validaci se nemusíme omezovat pouze na použití jednoho z výše uvedených jazyků. Naopak velký potenciál může být skryt v kombinaci dvou či více jazyků. Taková kombinace by mohla zajistit maximální vyjadřovací schopnost při zachování určité přehlednosti a spravovatelnosti definic. Ideálním kandidátem na kombinování je Schematron. Jeho architektura přímo nabádá, aby se stal doplňkovým jazykem k jinému validačnímu jazyku. Většina struktury tak může být vyjádřena klasickým způsobem. Špatně vyjádřitelná či klasicky nevyjádřitelná omezení lze definovat dodatečně pomocí Schematronu.

Příklad 2.6. Příklad Schematronu vnořeného do Relax NG

Schematron zde dodefinovává dodatečné omezení pro element `<a>` deklarovaný v Relax NG.

```
<define name="a">
  <element name="a">
    <!-- Začátek Schematronového fragmentu -->
    <sch:pattern name="nested.a">
      <sch:rule context="html:a">
        <sch:report test="descendant::html:a">
         Element a nesmí obsahovat sebe sama na libovolné úrovni vnoření.
        </sch:report>
      </sch:rule>
    </sch:pattern>
```

```
      <!-- Konec Schematronového fragmentu -->
      <ref name="a.attlist"/>
      <ref name="Inline.model"/>
    </element>
  </define>
```

Relax NG i XML Schema nabízejí dobrou podporu vnořených Schematronových testů. V XML Schema lze Schematron ukrýt ve značkách určených pro anotace. Obsah elementu `<appinfo>` je při validaci ignorován. Vhodnou transformací je pak možné vybrat pouze Schematronové definice a validovat oproti nim. Problematické je, že anotační elementy není možné přidat na libovolné místo definice. Ještě lepší mechanizmus nabízí Relax NG. Ten totiž ignoruje jakoukoli součást definice zapsanou v jiném jmenném prostoru, než je jmenný prostor používaný tímto jazykem. Schematron tak můžeme přidávat prakticky kamkoli. Dodatečná omezení týkající se určitého aspektu definice tak mohou být zapsána přímo tam, kde jsou definována i související omezení v rámci Relax NG. Validaci vůči Schematronu lze opět uskutečnit po jednoduché transformaci.

Otázkou je, jakým způsobem by mělo smysl kombinovat DSD. Tento jazyk stojí někde uprostřed dvou rozdílných přístupů k validaci XML dokumentů. Sám o sobě má silnou vyjadřovací schopnost v obou těchto směrech. Nedosahuje ovšem takových výsledků jako kombinace Schematronu a jiného jazyka. Kombinace se Schematronem ani s Relax NG nebo XML Schema by tento problém nevyřešila. Obě zmíněné možnosti by DSD nepřinesly žádnou přidanou hodnotu. Jazyky by se svými schopnostmi vždy z velké části překrývaly. Definice by se staly zbytečně složité a špatně spravovatelné.

Validaci oproti více validačním jazykům lze definovat i pomocí tzv. NRL (*Namespace Routing Language*) viz. [NRL]. Tento jazyk slouží k mapování různých jmenných prostorů na definice omezení psané v různých validačních jazycích. Fragmenty XML v příslušném jmenném prostoru jsou pak validovány oproti specifikovaným definicím. Ovšem pro jeden jmenný prostor lze definovat i více validačních definic. Následující příklad nám ukazuje, jakým způsobem lze konfigurovat validaci XHTML dokumentů oproti Relax NG i Schematronové definici zároveň.

Příklad 2.7. NRL

```
<rules xmlns="http://www.thaiopensource.com/validate/nrl">
  <namespace ns="http://www.w3.org/1999/xhtml">
    <validate schema="xhtml.rng"/>
    <validate schema="xhtml.sch"/>
  </namespace>
</rules>
```

2.6. Závěr

Po důkladném zvážení všech pro a proti se nejvýhodnější kombinací validačních jazyků pro doplňkovou validaci HTML a XHTML dokumentů zdá být kombinace Relax NG

a Schematronu. Tato kombinace přináší výbornou vyjadřovací schopnost (lze vyžít regulárních výrazů i jazyka XPath) a velice dobrou integrovatelnost obou jazyků. Integrovaný Schematron lze jednoduše validovat pomocí běžných transformačních nástrojů v rámci XML. Není tedy potřeba využívat dodatečné nástroje, jako je NRL. Ty by mohly mít význam v případě kombinované validace vůči mnoha různým validačním jazykům nebo při validaci různých vnořených XML fragmentů s různými jmennými prostory.

Velice silnou kombinací by jistě bylo i XML Schema spolu se Schematronem. Dá se říci, že vyjadřovací schopnost této a předešlé varianty je téměř totožná. Pro Relax NG ovšem mluví lepší čitelnost definic, lepší (i když téměř srovnatelná) integrovatelnost se Schematronem, dobře navržené možnosti modularity, ale hlavně obecně jednoduchost a jistá elegance vyjadřování.

DSD je podle mého názoru jazyk s velkým potenciálem a výbornou vyjadřovací schopností. Proti mluví určitá nedospělost, která se vyznačuje velkými změnami ve struktuře jazyka ve verzi dvě oproti předešlé verzi jedna. Tento jazyk také není tak široce používán jako jeho konkurenti. To vede k horší podpoře a nedostatku kvalitních nástrojů. Nasazení tohoto jazyka je tedy rizikové. Neexistuje zde dostatečná komunita uživatelů, která může pomoci řešit případné problémy. Použití pouze jednoho validačního jazyka je vzhledem k jednoduchosti definice lákavé, ovšem zmíněné kombinace Schematronu a jiného jazyka jsou velice dobře integrovatelné a zároveň i čitelnost je na takové úrovni, že v obecném kontextu ztrácí nasazení DSD pro doplňkovou validaci v rámci této práce smysl.

Samotné Relax NG nebo XML Schema přinášejí už samy o sobě výrazně lepší vyjadřovací schopnost než DTD. Jenom oblast datových typů řeší celou řadu omezení, které nebylo možné definovat pomocí DTD. Ovšem kombinace se Schematronem přináší poměrně velkou přidanou hodnotu. I v případě, že lze určité omezení formalizovat bez využití Schematronu, může být tato definice poměrně složitá a její zakomponování do zbytku definice může vyžadovat značné změny na více místech. V takovém případě přináší Schematron jednoduchou a čitelnou alternativu „za rozumnou cenu“. Jeho přidáním se totiž nezhoršuje nijak zásadně komplexnost definice ani čitelnost (spíše naopak). Jedinou komplikací je problém správného řešení vícejazyčné validace. Ovšem s dobře navrženou validační aplikací se i tento problém stává jednoduše řešitelným.

Ještě jsme se nevěnovali kombinaci XML DTD a Schematronu. Tato možnost by znamenala velice jednoduchou implementaci. Stačilo by užít stávající definice a přidat pouze Schematronové testy pro chybějící omezení. Jistě by šlo o řešení velice jednoduché, které by přinášelo neoddiskutovatelnou přidanou hodnotu, ale mnoho omezení by se nedalo řádně podchytit. Hlavně simulovaná funkcionalita datových typů ve Schematronu není dostatečně výstižná, aby podchytila všechny rozdílné požadavky kladené standardy HTML. XPath ve verzi 1.0 se v tomto ohledu nemůže rovnat vyjadřovacím schopnostem regulárních výrazů podporovaných v jiných validačních jazycích.

Zajímavá je jistě i možnost kombinace XML DTD a DSD užitého pouze pro doplňkovou validaci. Ta by oproti předešlé alternativě přinášela větší expresivitu v oblasti datových typů a menší v oblasti popisu struktury. Hlavní nevýhodu tohoto řešení lze vidět v malé rozšířenosti DSD a špatné integrovatelnosti obou jazyků.

Tabulka 2.3. Souhrnný přehled schopností kombinací validačních jazyků.

Vlastnost	XML DTD + Schematron	XML Schema + Schematron	Relax NG + Schematron	DSD
Obecné vlastnosti				
XML syntaxe	ANO	ANO	ANO	ANO
Podpora jmenných prostorů	NE	ANO	ANO	ANO
Datové typy				
Zabudované datové typy	10	37	2	0
Rozšiřitelnost datových typů	Částečná	ANO	ANO	ANO
Atributy				
Vylučovací výskyt atributů	ANO	ANO	ANO	ANO
Podmíněná hodnota atributu	ANO	ANO	ANO	ANO
Podmíněný výskyt atributu na základě hodnoty jiného atributu	ANO	ANO	ANO	ANO
Povinné a nepovinné atributy	ANO	ANO	ANO	ANO
Elementy				
Různé typy obsahu elementů	ANO	ANO	ANO	ANO
Pevné pořadí elementů	ANO	ANO	ANO	ANO
Neřazené sekvence elementů	Částečně (nutné definovat všechny možnosti)	ANO	ANO	ANO
Exklusivita (jeden vylučuje druhý)	ANO	ANO	ANO	ANO
Počet výskytu elementů	ANO	ANO	ANO	ANO
Podmíněný výskyt elementů	ANO	ANO	ANO	ANO
Modularita				
Zahrnutí externích definic	ANO	ANO	ANO	ANO
Dědění jednoduchých datových typů restrikcí	ANO	ANO	ANO	ANO
Dědění jednoduchých datových typů expanzí	NE	NE	NE	NE
Dědění definic restrikcí	ANO	ANO	ANO	ANO
Dědění definic expanzí	NE	ANO	NE	NE
Přepis definic	NE	NE	ANO	NE

Kapitola 3

Formalizace omezení pomocí vybrané kombinace validačních jazyků

3.1. Úvod

V této kapitole se budeme zabývat formalizací jednotlivých omezení kladených na HTML dokumenty pomocí jazyků Relax NG a Schematron. Tato kapitola popisuje nejen zajímavé aspekty konkrétní implementace některých omezení, ale i celkovou architekturu definic. Popisuje i metodiku implementačních prací a nastiňuje základní vývojový cyklus, který pomáhá zajistit konzistenci a správnost definic. Pokud nebude řečeno jinak, budeme se zabývat výhradně formalizací omezení specifikovaných v doporučení W3C pro jazyky HTML 4.01 a XHTML 1.0. K formalizaci směrnic WCAG 1.0 se dostaneme v závěru této kapitoly.

3.2. Struktura definice

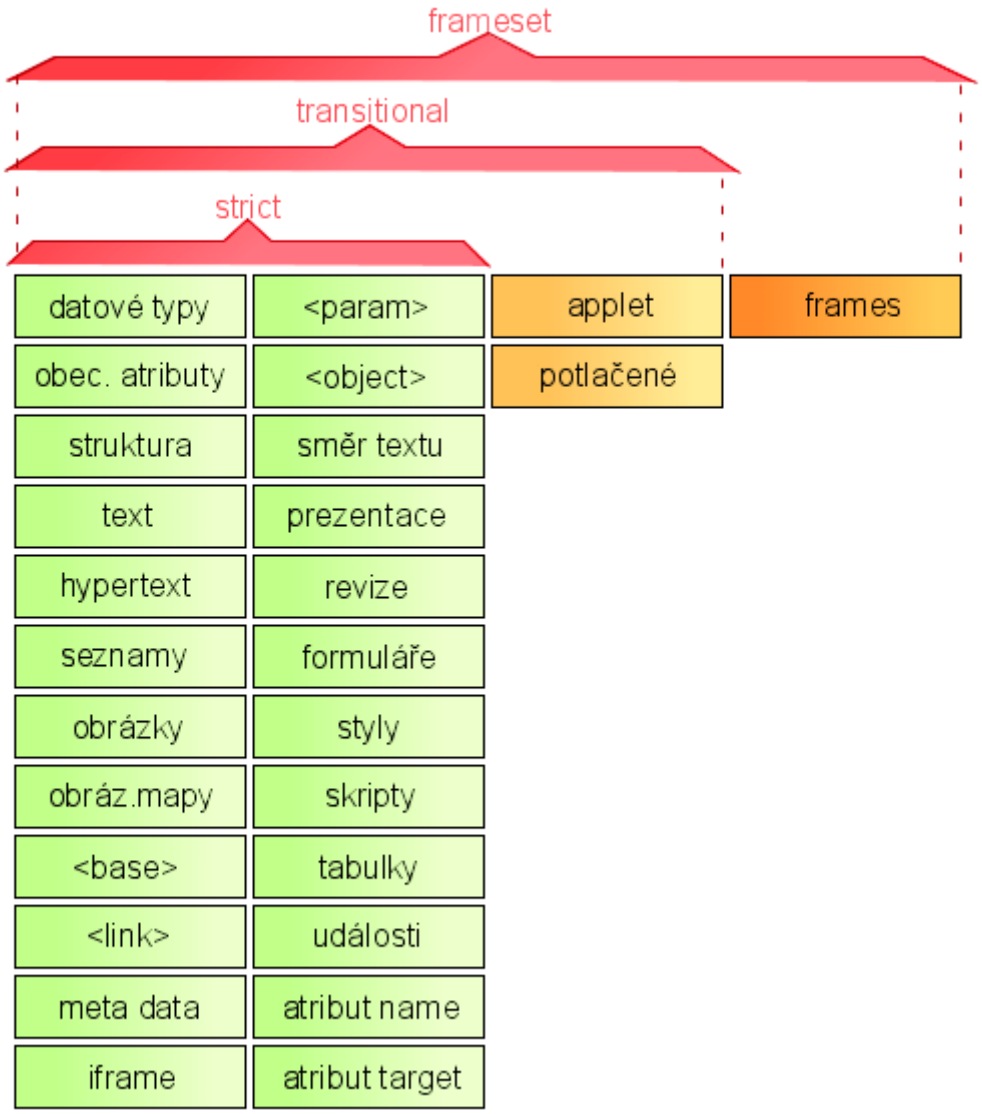
Jako základ pro implementaci omezení byla zvolena modulární definice XHTML napsaná Jamesem Clarkem viz. [XHTMLMOD-RNG]. Ta vychází z modularizace XHTML popsané v doporučení W3C viz. [XHTMLMOD]. Toto doporučení zatím nebylo v této práci zmíněno. Ve stručnosti se dá říci, že modularizace XHTML vychází plně z XHTML 1.0, ale nerozlišuje již mutace jazyka *strict*, *transitional* a *frameset*. Místo toho zavádí mnoho malých modulů, z nichž každý definuje určitou jasně vymezenou oblast jazyka.

Takový přístup přináší řadu výhod. Definice jsou přehledné a neopakují se na více místech. Navíc modularizace umožňuje vytvářet mnoho různých kombinací podmnožin jazyka HTML a validovat pouze tyto podmnožiny. Validace takovýchto podmnožin je nad rámec této práce. Ovšem dodržení modularity definic (tak jak je popsána v [XHTMLMOD]) přinese hlavně jejich lepší čitelnost a spravovatelnost. Správným poskládáním příslušných modulů bude možné vytvořit jednotlivé mutace jazyka HTML podle [HTML4] a [XHTML1].

Použitá definice od Jamese Clarka je v zásadě implementací modularizace XHTML v Relax NG. Oproti implementaci od W3C, která využívá XML DTD, demonstruje tato definice jednoduchost, s jakou lze vytvářet a spojovat jednotlivé moduly v Relax NG. Zatímco v DTD je nutná existence jakési společné definice, do které lze teprve zahrnovat další moduly, v Relax NG takovouto definici nepotřebuje. Jednotlivé moduly se po svém zahrnutí samy umí postarat o správnou modifikaci definic příslušných elementů.

Zmíněná implementace definic posloužila této práci pouze jako hrubý základ. Vzhledem k tomu, že se jedná o čistě ukázkovou záležitost, která slouží k demonstraci některých vlastností jazyka Relax NG, obsahují tyto definice velké množství nepřesností a chyb, které bylo nutné v prvé řadě odstranit. Moduly jsou navíc navrženy tak, že není možné oddělit definice pro mutace *transitional* a *frameset*. Dalším úkolem tedy bylo fragmentovat definice, aby byla umožněna separátní validace těchto mutací.

Původní definice v Relax NG si nekladla za cíl žádnou doplňkovou validaci. Záměrem bylo pouze vyjádření modularity XHTML elegantněji, než je to možné v DTD. Posledním úkolem tedy bylo doplnit všechna dodatečná omezení (nad rámec DTD), která lze vyjádřit díky skvělé expresivitě kombinace validačních jazyků Relax NG a Schematron.



Obrázek 3.1. Jednotlivé moduly definice a vyjádření mutací jazyka HTML pomocí jejich uskupení

Moduly využité v definici odpovídají zhruba modulům definovaným v [XHTML-MOD]. V některých případech jsou ještě o něco málo podrobnější, než uvádí toto doporučení W3C. Ačkoli vyjádření různých podmnožin HTML (včetně specifické mutace zvané XHTML Basic) je pomocí různých kombinací modulů principielně možné, tato práce se bude snažit o korektní validaci pouze v případě XHTML 1.0 a HTML 4.01. Validace jiných mutací jazyka HTML je sice nad rámec této práce, ovšem zvolená architektura dává výborný odrazový můstek k budoucímu rozšíření definice tak, aby kompletně splňovala doporučení pro modularizaci XHTML.

3.3. Metodika zápisu formalizací – vývojový cyklus

Vývojový cyklus implementace formalizací jednotlivých omezení, který byl aplikován v této práci, ukazuje obrázek 3.2 – „Vývojový cyklus při formalizaci omezení“. V první fázi je vybráno konkrétní omezení, které má být naimplementováno. Podkladem pro tento výběr je kapitola 1 – „*Analýza omezení a požadavků kladených na HTML a XHTML dokumenty*“. V druhé fázi je pak zvolen vhodný způsob formalizace. Cílem je vybrat logické začlenění do zbytku definic a vhodný modul, jehož se toto omezení stane součástí. Je také nutné zvolit jazyk, který bude nejlépe odpovídat typu omezení a jeho specifikám. V našem případě tedy volíme mezi Relax NG a Schematronem.

Následuje implementace vybraného omezení. V této fázi je formalizované omezení zapsáno ve vybraném jazyce a umístěno na správné místo v některém z modulů. Součástí této fáze je i tvorba výstižných komentářů k implementovaným definicím. Tyto komentáře jsou obzvláště důležité u Schematronových testů, kde by měly důsledně popisovat mechanizmus testu, a u regulárních výrazů, kde by měly sloužit jako slovní popis takového výrazu.

Poslední fáze, která se zabývá testováním, je fází nejdůležitější. Každé implementované omezení je nutné řádně otestovat. Je nutné zjistit, zda se nově implementované doplňkové omezení chová tak, jak jsme skutečně zamýšleli. Důležitou součástí takového testu je i ověření, zda definice zůstávají v souladu s oficiálním DTD od W3C. Všechny chyby, které je možné automaticky validovat proti tomuto DTD, by měla odhalit jako chyby i naše definice.

Pokud pomineme implementaci jednotlivých definic, je výstupem vývojového cyklu v každé jeho fázi i soustava testových úloh. Každá taková úloha má v podstatě podobu jednoduchého HTML dokumentu. Tento dokument může být validní i chybný. Cílem testu je určit, zda jsou validní dokumenty skutečně validní a zda jsou v chybných dokumentech správným způsobem označeny všechny chyby. Každá testová úloha se váže ke konkrétnímu omezení. Implementace každého nového omezení má vyústit v napsání jedné, nebo několika testových úloh, které prověřují důležité aspekty tohoto omezení. V průběhu vývojového cyklu takto vzniká hierarchicky uspořádaná knihovna testových úloh. Takováto knihovna přináší mnoho výhod. Umožňuje například udržovat definice konzistentní v průběhu celého vývojového cyklu. Součástí testovací fáze je totiž i kontrola toho, zda nebyla přidáním nové definice narušena nějaká již implementovaná funkcionalita a zda se všechny přítomné testové úlohy chovají stále správně a konzistentně s DTD. Toho je dosahováno tím, že není testováno pouze nové omezení a jeho testové úlohy, ale při každém zásahu do definice dojde ke kompletnímu testu

Obrázek 3.2. Vývojový cyklus při formalizaci omezení

všech testových úloh v knihovně. Teprve pokud se všechny testy chovají korektně, lze definici v aktuální podobě zachovat.

Takováto automatizovaná podpora vývoje je možná díky implementaci testového prostředí, jehož popis je součástí kapitoly 4 – *„Implementace prototypové validační aplikace“*.

3.4. Příklady implementace některých formalizací

V této části se podíváme na to, jak je konkrétním způsobem řešena formalizace některých omezení. Pokusíme se zvolit k tomuto účelu vhodné kandidáty, kteří budou demonstrovat různé formalizační přístupy. Kompletní výčet formalizací opatřených komentáři je součástí přiložené prototypové validační aplikace a není tedy plně zmiňován v tomto textu.

3.4.1. Datové typy

Ve specifikaci HTML je celá řada různých omezení datových typů, které nelze podchytit v DTD. Tato omezení lze v zásadě velmi dobře formalizovat v Relax NG pomocí knihovny datových typů jazyka XML Schema. V některých jednoduchých případech je možné použít přímo některý ze zabudovaných datových typů. Sem patří i jednotný identifikátor zdrojů (URI). Pokud v definici užijeme datový typ anyURI, je pak vyžadována přesná syntaxe identifikátoru podle [RFC2396]. Následující příklad ukazuje definici takového typu v Relax NG.

Příklad 3.1. Datový typ anyURI

```
<define name="URI.datatype">
  <data type="anyURI"/>
</define>
```

Ne vždy však vystačíme se zabudovanými datovými typy. V takovém případě můžeme aplikovat další omezení. Jde o různé dodatečné parametry a ve složitějších případech o použití regulárních výrazů. Například požadavek standardu, aby byl atribut `tabindex` kladným celým číslem v rozmezí 0 až 32767, lze velmi dobře formalizovat následujícím způsobem.

Příklad 3.2. Omezení atributu `tabindex`

```
<define name="tabindexNumber.datatype">
  <data type="nonNegativeInteger">
    <!-- pouze sekvence číslic -->
    <param name="pattern">[0-9]+</param>
    <param name="minInclusive">0</param>
    <!-- shora omezeno, maximálně 32767 včetně -->
    <param name="maxInclusive">32767</param>
  </data>
</define>
```

V příkladu vyžíváme zabudovaný datový typ nonNegativeInteger. Ten dále omezujeme, protože tak jak je specifikován v [XMLSCH-DT] může obsahovat i nežádoucí znaménko „+“. Je tedy potřeba využít velmi jednoduchého regulárního výrazu `[0-9]+` a omezit tvar pouze na sekvenci číslic. Horní omezení je zavedeno pomocí parametru `maxInclusive`.

Některé regulární výrazy použité v této práci jsou inspirované dokumentem [XHTML-XMLSCH]. Tento dokument dává k dispozici ukázkovou implementaci omezení kladených na jazyk XHTML 1.0. Tato implementace využívá validační jazyk XML Schema. Tento dokument je ve fázi rozpracování. Je tedy čistě orientační a nemá normativní charakter. Tomu odpovídají i některé chyby v definicích, a dokonce i v regulárních výrazech. Přesto se některé výrazy ukázaly jako užitečné pro tuto práci a byly do ní zahrnuty. Jiné naopak potřebovaly doplnit, opravit a nebo bylo jednoduše možné dosáhnout lepšího efektu s využitím Schematronu nebo Relax NG.

Někdy je třeba definovat značně složitá omezení s různými podmíněnými výskyty některých tvarů. Regulární výrazy jsou tím pravým nástrojem pro tento úkol. Jejich expresivita je relativně vysoká. Například regulární výraz `[A-Za-z]+|#[0-9A-Fa-f]{6}` omezuje barvy využívané v některých atributech HTML (hlavně v těch potlačených). Tento regulární výraz umožňuje zapisovat barvy v hexadecimálním kódu, nebo jako sérii malých a velkých písmen. Specifikace ovšem normativně stanovuje seznam šestnácti barev, které lze využít v jejich jmenném tvaru. Regulární výraz by potom vypadal takto:

```
(Black|Green|Silver|Lime|Gray|Olive|White|Yellow|Maroon|Navy|Red|Blue|
Purple|Teal|Fuchsia|Aqua)|#[0-9A-Fa-f]{6}
```

Specifikace ale uvádí, že názvy barev mohou být zapsány nezávisle na velikosti písmen. Regulární výraz je tedy nutné upravit následujícím způsobem.

```
[bB][lL][aA][cC][kK]|[gG][rR][eE][eE][nN]|[sS][iI][lL][vV][eE][rR]|[lL]
[iI][mM][eE]|[gG][rR][aA][yY]|[oO][lL][iI][vV][eE]|[wW][hH][iI][tT][eE]
|[yY][eE][lL][lL][oO][wW]|[mM][aA][rR][oO][oO][nN]|[nN][aA][vV][yY]|
[rR][eE][dD]|[bB][lL][uU][eE]|[pP][uU][rR][pP][lL][eE]|[tT][eE][aA]
[lL]|[fF][uU][cC][hH][sS][iI][aA]|[aA][qQ][uU][aA]|[0-9A-Fa-f]{6}
```

Tento výraz je poměrně nečitelný, ale na druhou stranu plně popisuje omezení kladená na vyjádření barev v HTML.

Dalším zajímavým regulárním výrazem, který je použit v definici, je výraz `_(blank|self|parent|top)|[A-Za-z].*`. Ten omezuje odkazy na rámy zapsané v atributech `target`. V souladu s doporučením HTML 4.01 viz. [HTML4] umožňuje pouze rezervovaným výrazům, aby začínaly podtržením. Jinak je jako první znak vyžadováno písmeno.

Zastavme se ve stručnosti ještě u dvou ilustrativních výrazů. Výraz `[-+]?(\d+|\d+(\.\d+)?%)|[1-9]?(\d+)?\*` omezuje atributy, které specifikují délky. S tímto omezením je možné vyjadřovat relativní, procentuální a absolutní délky pouze tak, jak je specifikuje HTML 4.01.

Výraz `[^,]+(,\s*[^,]+)*` omezuje seznamy médií. Autoři smějí do takových seznamů zapsat v podstatě libovolná jména. Břímě odpovědnosti za taková jména leží na prohlížečích. Ty mají neznámá jména ignorovat. Proto tento regulární výraz kontroluje pouze to, zda se jedná o jednu položku, nebo o seznam položek oddělených čárkou. Takový seznam pak například nesmí čárkou začínat.

Některé atributy HTML mohou obsahovat seznamy, které jsou odděleny mezerou (nikoli čárkou). Takové seznamy lze vyjádřit speciálním konstruktem jazyka Relax NG. Není tedy třeba využívat špatně čitelných regulárních výrazů. Následující příklad ukazuje atribut, který může obsahovat seznam identifikátorů URI oddělených vzájemně mezerou.

Příklad 3.3. Datový typ URIs může obsahovat URI, nebo jejich seznam se členy oddělenými mezerou

```
<define name="URIs.datatype">
  <list>
    <oneOrMore>
      <data type="anyURI"/>
    </oneOrMore>
  </list>
</define>
```

Pomocí Relax NG lze mnohdy elegantně vyjádřit i složitější závislosti v oblasti datových typů. Následující příklad je sice možné částečně vyjádřit i ve Schematronu, ale zápis v Relax NG je elegantnější, čitelnější a úplnější. Využívá podmíněného výskytu atributu v kombinaci s regulárními výrazy. Specifikace ukládá zvláštní požadavky na atribut `coords` elementu `<area>`. Jeho hodnota může nabývat různé formy, podle toho jaký tvar je tomuto elementu nastaven. Například pokud se jedná o obdélník, musí být atribut `coords` naplněn čtyřmi absolutními, nebo procentuálními délkami oddělenými čárkou. Jiný tvar pak souřadnice mají v případě kruhu nebo polygonu. Podívejme se společně, jakým způsobem lze takové omezení implementovat.

Příklad 3.4. Tvar souřadnic elementu `<area>` v závislosti na hodnotě atributu `shape`

```
<define name="shape.attrib">
  <optional>
    <choice>
      <group>
        <attribute name="coords">
         <!-- Implicitním tvarem je obdélník. Pokud není přítomen atribut ▶
shape, předpokládají se tedy souřadnice pro obdélník (rect). -->
          <ref name="coords.rect.datatype"/>
        </attribute>
      </group>
      <group>
       <!-- V případě hodnoty default je tvarem celá plocha a není tedy ▶
třeba specifikovat jednotlivé souřadnice. -->
        <attribute name="shape">
          <value>default</value>
        </attribute>
      </group>
      <group>
        <attribute name="shape">
          <value>poly</value>
        </attribute>
        <attribute name="coords">
          <ref name="coords.datatype"/>
        </attribute>
      </group>
      <group>
        <attribute name="shape">
          <value>rect</value>
        </attribute>
        <attribute name="coords">
          <ref name="coords.rect.datatype"/>
        </attribute>
      </group>
      <group>
```

```
          <attribute name="shape">
            <value>circle</value>
          </attribute>
          <attribute name="coords">
            <ref name="coords.circle.datatype"/>
          </attribute>
        </group>
      </choice>
    </optional>
  </define>

  <!-- zde jsou definovány různé podoby povolených souřadnic pro různé typy ▶
  tvarů -->
  <define name="coords.datatype">
    <data type="string">
      <!-- tento tvar souřadnic je povolen v případě polygonu -->
      <!-- Regulární výraz umožňuje zapsat libovolný počet délek oddělených ▶
  čárkou. -->
      <param name="pattern">( )*([-+]?(\d+|\d+(\.\d+)?%))(( )*,( ▶
  )*[-+]?(\d+|\d+(\.\d+)?%))*( )*</param>
    </data>
  </define>
  <define name="coords.circle.datatype">
    <data type="string">
      <!-- tento tvar souřadnic je povolen v případě kruhu -->
      <!-- je možné zapsat přesně tři délky oddělené čárkou -->
      <param name="pattern">( )*([-+]?(\d+|\d+(\.\d+)?%)( )*,( ▶
  )*){2}([-+]?(\d+|\d+(\.\d+)?%))( )*</param>
    </data>
  </define>
  <define name="coords.rect.datatype">
    <data type="string">
      <!-- tento tvar souřadnic je povolen v případě obdélníku -->
      <!-- je možné zapsat přesně čtyři délky oddělené čárkou -->
      <param name="pattern">( )*([-+]?(\d+|\d+(\.\d+)?%)( )*,( ▶
  )*){3}([-+]?(\d+|\d+(\.\d+)?%))( )*</param>
    </data>
  </define>
```

Následující ilustrativní příklad pro srovnání ukazuje možnou formalizaci omezení tvaru souřadnic pro kruh pomocí Schematronu. Využívá práci s řetězci a jejich převod na čísla. Takovýto test neumožňuje zapisovat souřadnice jako procenta. Podobným způsobem lze formalizovat i omezení pro obdélníkový tvar.

Příklad 3.5. Tvar souřadnic elementu `<area>` v závislosti na hodnotě atributu `shape`

```
<sch:pattern name="area.shape.coords">
  <sch:rule context="html:area">
    <sch:report test="@shape = 'circle' and ▶
(string(number(substring-before(@coords,','))) = 'NaN' or ▶
string(number(substring-before(substring-after(@coords,','),','))) = 'NaN' ▶
or string(number(substring-after(substring-after(@coords,','),','))) = ▶
'NaN')">
      Pokud je atribut shape elementu area nastaven na hodnotu circle,
      atribut coords musí obsahovat právě 3 čísla oddělená čárkou.
    </sch:report>
  </sch:rule>
</sch:pattern>
```

3.4.2. Skládání definic

Mutace jazyka HTML *transitional* přidává řadu dodatečných atributů, které není možné používat, pokud píšeme dokument v mutaci *strict*. Pojďme se podívat, jakým způsobem jsou v Relax NG obohaceny již definované elementy o tyto dodatečné atributy. Toto obohacení probíhá automaticky pouhým přidáním dodatečných modulů.

Způsob implementace tohoto mechanizmu si ukážeme na elementu `<ul>`, který značkuje položky seznamů. Tento element je definován v modulu seznamů, který je součástí mutace *strict*, *transitional* i *frameset*. Tuto definici vidíme v následujícím příkladu.

Příklad 3.6. Definice elementu `<ul>` v modulu seznamů

```
<define name="ul">
  <element name="ul">
    <ref name="ul.attlist"/>
      <oneOrMore>
        <ref name="li"/>
      </oneOrMore>
  </element>
</define>

<define name="ul.attlist">
  <ref name="Common.attrib"/>
</define>
```

Vidíme, že element `<ul>` může obsahovat elementy `<li>` a skupinu atributů, které jsou společné i mnoha jiným elementů. Mutace *transitional* a *frameset* ovšem rozeznává ještě celou řadu dalších atributů tohoto elementu. Do některého modulu, který je zahrnut pouze v případě validace těchto mutací, je třeba přidat následující definici.

Příklad 3.7. Definice dodatečných atributů elementu `<ul>`

```
<define name="ul.attlist" combine="interleave">
  <optional>
    <attribute name="type">
      <choice>
        <value>disc</value>
        <value>square</value>
        <value>circle</value>
      </choice>
    </attribute>
  </optional>
  <optional>
    <attribute name="compact">
      <value>compact</value>
    </attribute>
  </optional>
</define>
```

Pokud validujeme dokument v mutaci *transitional*, jsou zahrnuty oba zmíněné moduly. Obě definice označené jako „ul.attlist“ jsou tedy sloučeny a validními se stávají i ty elementy `<ul>`, které obsahují atribut `compact` nebo `type`.

Takto nám Relax NG umožňuje jednoduše rozšiřovat definice elementů v závislosti na použitých modulech. Ovšem ne všechny expanze lze vyjádřit tak snadno. Například pokud má dodatečný modul učinit povinný atribut nepovinným. V tomto případě nabízí Relax NG o něco méně elegantní, ale velmi silný nástroj, jak takové expanze dosáhnout. V místě, ve kterém zahrnujeme inkriminovaný modul, je třeba příslušnou definici kompletně přepsat. Následující příklad ukazuje, jakým způsobem lze konkrétní atribut učinit volitelným, pokud zahrnujeme méně striktní modul.

Příklad 3.8. Expanzivní přepis

Atribut `href` elementu `<base>` je povinný v mutaci *strict* a nepovinný v mutaci *transitional*. Celá definice skupiny atributů tohoto elementu je přepsána.

```
<include href="modules/base.rng">
  <!-- přepis -->
  <define name="base.attlist">
    <optional>
      <attribute name="href">
      <ref name="URI.datatype"/>
      </attribute>
    </optional>
  </define>
</include>
```

Podobný přístup je třeba aplikovat i v případě, že potřebujeme přidáním modulu některé elementy naopak omezit. Restrikce lze dosáhnout i použitím vnořeného Schematronu. Například je zajímavé, že atribut `name` elementu `<param>` je nepovinný pouze v mutaci *strict*. Moduly ostatních mutací tedy musí obsahovat dodatečnou restrikci. Její vyjádření ve Schematronu ukazuje následující příklad.

Příklad 3.9. Dodatečná restrikce zapsaná ve Schematronu

```
<sch:rule context="html:param">
    <sch:assert test="@name">
      Atribut name elementu param je povinný.
    </sch:assert>
  </sch:rule>
```

3.4.3. Využití Schematronu

Kromě modulárních restrikcí je Schematron využíván všude tam, kde nelze některá omezení vyjádřit pomocí Relax NG. To ovšem není jediné kriterium jeho nasazení. Někdy je formalizace pomocí Schematronového testu mnohem elegantnější a vyžaduje menší rozsah zásahu do definice než využití Relax NG. V této části si ukážeme několik ilustrativních případů využití Schematronu v definici.

Specifikace HTML 4.01 nedovoluje některým elementům, aby obsahovaly samy sebe na libovolné úrovni vnoření. Tento fakt lze velice snadno vyjádřit pomocí Schematronového testu. Následující příklad ukazuje nejen tento test, ale i obecně způsob vnoření Schematronu do Relax NG. Velkou výhodou je, že v Relax NG je možné psát Schematronové testy úplně kamkoli (díky odlišnému jmennému prostoru), tedy i přímo do definic elementů, kterých se týkají.

Příklad 3.10. Dodatečná restrikce zapsaná v Schematronu

V tomto příkladu je dobré povšimnout si toho, jakým způsobem je deklarován Schematronový jmenný prostor v elementu `<grammar>`. Dále je důležité povšimnout si definice jmenného prostoru XHTML. Tato definice je nezbytná. Bez ní nelze validovat XHTML dokumenty, které lze očekávat právě v tomto jmenném prostoru. Prefix „html“ poté slouží ve výrazech XPath pro adresování jednotlivých elementů z tohoto jmenného prostoru. Například element `<form>` je nutné adresovat jako „html:form“.

```
<!-- implicitní jmenný prostor patří Relax NG, ale jmenný prostor označený ▶
předponou "sch" patří Schematronu -->
<grammar xmlns="http://relaxng.org/ns/structure/1.0" ▶
xmlns:sch="http://www.ascc.net/xml/schematron">

<!-- zde je definován jmenný prostor pro xhtml, což je důležité pro ▶
adresování HTML elementů pomocí XPath. -->
<sch:ns prefix="html" uri="http://www.w3.org/1999/xhtml"/>
```

```
<!-- definice elementu form v Relax NG -->
<define name="form">
  <element name="form">
    <!-- začátek vnořeného Schematronu -->
    <sch:pattern name="nested.form">
      <sch:rule context="html:form">
        <sch:report test="descendant::html:form">
          Element form nesmí obsahovat žádné další elementy form na ▶
libovolné úrovni vnoření.
        </sch:report>
      </sch:rule>
    </sch:pattern>
    <!-- konec vnořeného Schematronu -->
    <ref name="form.attlist"/>
    <oneOrMore>
      <ref name="Block.class"/>
    </oneOrMore>
  </element>
</define>

</grammer>
```

Specifikace HTML 4.01 dále vyžaduje, aby atributy `name` a `id` měly stejnou hodnotu, pokud jsou v případě jednoho elementu použity společně. Tento požadavek nelze v našem případě vyjádřit jinak než Schematronem. Následující příklad ukazuje zápis takového omezení. Tato formalizace je zajímavá i tím, že se vztahuje obecně na všechny elementy. Ty jsou vybrány jako „html:*“.

Příklad 3.11. Dodatečná restrikce zapsaná ve Schematronu

```
<sch:rule context="html:*">
  <sch:report test="@id and @name and @id != @name">
    Atributy id a name musí mít stejnou hodnotu, pokud se vyskytují ▶
současně.
  </sch:report>
</sch:rule>
```

V předcházející části jsme řešili problematiku šestnácti povolených názvů barev. Regulární výraz byl poměrně složitý a špatně čitelný. Pojďme se teď podívat, jak by vypadalo vyjádření pomocí Schematronu. Funkce `translate` mění všechna malá písmena na velká, a umožňuje tak kontrolovat různé varianty názvů barev.

Příklad 3.12. Kontrola jmen barev

```
<define name="Color.datatype">
  <data type="string">
    <!-- regulární výraz pro kontrolu číselné formy -->
    <param name="pattern">[A-Za-z]+|#[0-9A-Fa-f]{3}|#[0-9A-Fa-f]{6}</param>
  </data>
  <!-- Následující Schematronový test kontroluje navíc jména barev, zda-li ▶
patří mezi 16 specifikovaných jmen. -->
  <sch:pattern name="color.datatype">
    <sch:rule context="html:font | html:basefont">
      <sch:report test="@color and not(starts-with(@color,'#'))
      and translate(@color, 'abcdefghijklmnopqrstuvwxyz', ▶
'ABCDEFGHIJKLMNOPQRSTUVWXYZ') != 'BLACK'
      and translate(@color, 'abcdefghijklmnopqrstuvwxyz', ▶
'ABCDEFGHIJKLMNOPQRSTUVWXYZ') != 'SILVER'

      ... všech 16 názvů barev ...

      and translate(@color, 'abcdefghijklmnopqrstuvwxyz', ▶
'ABCDEFGHIJKLMNOPQRSTUVWXYZ') != 'TEAL'
      and translate(@color, 'abcdefghijklmnopqrstuvwxyz', ▶
'ABCDEFGHIJKLMNOPQRSTUVWXYZ') != 'AQUA'
      ">
        Pouze 16 jmen barev je rozeznáváno:
        Black, Green, Silver, Lime, Gray, Olive, White,
        Yellow, Maroon, Navy, Red, Blue, Purple, Teal,
        Fuchsia and Aqua.
      </sch:report>
    </sch:rule>
  </sch:pattern>
</define>
```

Předchozí příklad je velice těžkopádný. Dá se říci, že je horší než příslušný regulární výraz. Navíc kontroluje pouze hodnoty atributu `color`. Podobnou definici je nutné opakovat ještě pětkrát pro všechny atributy využívající datový typ „Color“. Regulární výraz je tedy pro popis tohoto omezení vhodnější, a proto je také použit v definici.

Budoucí verze Schematronu (ISO Schematron) umožní zapisovat podobné definice elegantněji pomocí podpory pro proměnné. Následující příklad ukazuje stejný test ve zjednodušené podobě.

Příklad 3.13. Proměnné v ISO Schematronu

Pomocí elementu `<let>` je translační funkce provedena pouze jednou a její výsledek je uložen do hodnoty proměnné *color*. Dále se pracuje pouze s touto proměnnou.

```
<sch:pattern name="color.datatype">
  <sch:rule context="html:font | html:basefont">
    <sch:let name="color" value="translate(@color, ▶
'abcdefghijklmnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
    <sch:report test="@color and not(starts-with(@color,'#')) and $color ▶
!= 'BLACK' .... and $color != 'AQUA'">
      Pouze 16 jmen barev je rozeznáváno.
    </sch:report>
  </sch:rule>
</sch:pattern>
```

Jednodušším příkladem užití Schematronu je následující test. Specifikace vyžaduje, aby určité typy ovládacích prvků obsahovaly hodnotu. Toto omezení platí pro prvky, které vyjadřují volbu z více alternativ. Je tedy nutné tyto alternativy nějakým způsobem označit. Proto není v následujícím příkladu v souladu s duchem specifikace vyžadována pouze přítomnost atributu `value`, ale i jeho naplnění nějakou hodnotou.

Příklad 3.14. Ovládací prvky *checkbox* a *radio* by měly mít neprázdnou hodnotu.

```
<sch:rule context="html:input">
  <sch:report test="((@type = 'radio' or @type = 'checkbox') and ▶
(not(@value) or string-length(@value) = 0))">
    Ovládací prvky radio a checkbox by měly mít specifikován neprázdný ▶
atribut value.
  </sch:report>
</sch:rule>
```

Silným nástrojem jazyka XPath jsou počty uzlů. Ty je možné získat pomocí funkce `count`. Díky této funkci lze potom řešit například omezení týkající se ovládacího prvku `<select>`, který je užíván pro výběry. Ten pracuje v zásadě ve dvou módech. Umožňuje více voleb současně (atribut `multiple`), a nebo pouze jednu. Jednotlivé možnosti ve výběru potom mohou být explicitně nastaveny jako vybrané (atribut `selected`). Více možností lze takto zvolit pouze v případě, že je nastaven atribut `multiple`. Podívejme se, jak lze toto omezení jednoduše formalizovat ve Schematronu.

Příklad 3.15. Využití funkce `count`

```
<sch:rule context="html:select">
  <sch:report test="not(@multiple) and count(html:option[@selected]) > 1">
    Elementy select bez atributu multiple nemohou mít více vybraných voleb.
  </sch:report>
</sch:rule>
```

Poslední zvolenou ilustrativní ukázkou využití Schematronu je zákaz atributu `usemap` pro obrázky vnořené v tlačítkách. Zajímavé je, že toto omezení platí na libovolné úrovni vnoření. Tento fakt lze formalizovat s využitím osy XPath s názvem *descendant*.

Příklad 3.16. Omezení atributů libovolně vzdálených subelementů

```
<sch:rule context="html:button">
  <sch:report test="descendant::html:img[@usemap]">
    Obrázky vnořené v elementech button nesmí obsahovat atribut usemap.
  </sch:report>
</sch:rule>
```

3.4.4. Úspěšnost formalizace

V zásadě se dá říci, že použitými metodami se podařilo formalizovat velkou většinu dodatečných normativních omezení, která jsou vyjádřena v sekci 1.3.3 – „Další omezení nad rámec DTD“. Formalizována nebyla ta omezení, která jsou příliš vágně definována, a jejich kontrola tedy vyžaduje lidský zásah. Takový je například požadavek nevyužívat tabulky pro rozvržení stránek nebo vytvářet vystihující popisky některých elementů. Dále nebyla formalizována ta omezení, která jsou sice automatizovaně kontrolovatelná, ale není možné je vyjádřit v použitých jazycích, případně by jejich vyjádření bylo příliš složité a rozsáhlé. Tato omezení shrnuje následující seznam.

- Elementy `<ins>` a `<del>` se mohou chovat jako řádkové i blokové v závislosti na kontextu.
- Nastavení implicitního skriptovacího nebo stylového jazyka v hlavičce dokumentu (ten může být totiž nastaven i později webovým serverem).
- Překrývající se buňky v tabulkách.
- Maximální počet sloupců v řádce tabulky a explicitně definovaný počet sloupců si musí odpovídat.
- Mezery by měly být zapisovány před počáteční značkou a po konečné značce (ne opačně).
- Není dovoleno užívat explicitní citační značky v elementu `<q>` (byla by potřeba kontrolovat přítomnost specifických citačních značek pro různé přirozené jazyky).

3.5. Formalizace omezení WCAG 1.0

Většina omezení vycházejících ze směrnic pro dostupnost obsahu webu je pro formalizaci příliš vágní. Směrnice 8, 13 a 14 se nepodařilo formalizovat vůbec. Ostatní směrnice jsou podchyceny částečně. Pro formalizaci omezení byl využit výlučně Schematron. Ten se na daný úkol ideálně hodí. Směrnice totiž nepopisují gramatiku nějakého jazyka, ale spíše jednotlivá omezení, která jsou na sobě v podstatě nezávislá.

Navíc jsou tato omezení mnohdy strukturálně velmi komplexní a těžko by šla formalizovat v jiném jazyce. Schematron je vhodný i z toho důvodu, že v něm lze jednoduše vyjádřit modulární restrikce, a to pouhým zapsáním pravidel do příslušného modulu. Formalizace se skládá ze 43 pravidel sloučených do skupin podle směrnice, které se týkají. Každý test obsahuje i informaci o prioritě viz. [WCAG1]. Kompletní seznam všech formalizovaných pravidel je k nalezení v příslušném modulu definice v rámci prototypové aplikace. V následujících odstavcích si ukážeme pouze výběr nejzajímavějších z nich.

V první směrnici se testová pravidla zabývají hlavně tím, zda jsou vyplněny příslušné atributy týkající se slovních popisů jednotlivých elementů. Jedná se hlavně o atributy `alt` a `longdesc`. Tato pravidla jsou většinou jednoduchá a není třeba je zde podrobněji popisovat.

Formalizace druhé směrnice se snaží předejít tomu, aby byla barva popředí a pozadí shodná. To by znamenalo nečitelnost textu. Zajímavé by jistě bylo i kontrolovat kontrast barev. Zde by bylo problematické využít formát RGB, ve které jsou barvy v HTML zapisovány. Kontrast jednotlivých složek spektra je totiž různý. Na zelenou je lidské oko o něco citlivější, naopak modrá se nám jeví většinou tmavší. Tento problém mohou vyřešit správně zvolené váhy, které budou tyto rozdíly brát v úvahu. Ovšem hodnoty vah se mohou lišit například podle druhu zobrazovacího zařízení. Problematický by byl i samotný převod hexadecimálních čísel na decimální pomocí XPath 1.0, ke kterému by bylo nutné přistoupit. Z těchto důvodů se nakonec test kontrastu v definici nevyskytuje. Není to ovšem velká ztráta, protože barevné informace by stejně měly být zachyceny ve stylových definicích.

Třetí směrnice se zabývá tím, aby nebyly užívány prezentační nástroje HTML (ty mají být definovány ve stylech). Tato směrnice navíc doporučuje užívání relativních délek místo absolutních. Testy jsou opět velice jednoduché. Kontrolují se výskyty prezentačních atributů a výskyty znaků „*“ a „%“ v atributech určujících délku.

Čtvrtá směrnice mimo jiné vyžaduje, aby se u prvního použití zkratky v dokumentu vyskytoval titulek s jejím vysvětlením. Následující příklad ukazuje využití Schematronu pro formalizaci takového požadavku. Chyba je ohlášena pro každou zkratku, která neobsahuje atribut `title` a ani neexistuje stejná předchozí zkratka, která by tento atribut obsahovala.

Příklad 3.17. Titulek musí být uveden minimálně u prvního výskytu zkratky

```
<sch:rule context="html:abbr">
  <sch:report test="not(@title) and not(preceding::html:abbr[. = ▶
string(current())][@title])">
    WCAG 1.0 Checkpoint 4.2 (Priority 3) Titulek musí být uveden minimálně ▶
u prvního výskytu zkratky.
  </sch:report>
</sch:rule>
```

Pátá směrnice se zabývá tabulkami. Formalizovány byly požadavky na popisky v tabulce a zkratky v záhlavích. Šestá směrnice vyžaduje využívání elementu `<noframes>` z důvodu kompatibility s prohlížeči, které nepodporují rámy. Toto omezení bylo jed-

noduše formalizováno. Ostatní omezení byla pro formalizaci již příliš vágní. Sedmá směrnice zakazuje nekontrolovatelné blikání a pohyb obsahu. Zde je kontrolována přítomnost proprietárních elementů `<blink>` a `<marquee>`, které mohou způsobovat právě takovéto chování obsahu. Tyto elementy jsou ovšem kontrolovány i tak, protože nejsou součástí definice jazyka HTML. Blikání a pohyb lze ovšem uskutečnit i některým skriptovacím jazykem. V tomto případě by byla kontrola mimořádně obtížná. Podobné je to i v případě požadavku, že by uživatel neměl být nekontrolovatelně přesměrováván na jiné dokumenty. V tomto případě lze testovat příslušný parametr v hlavičce dokumentu (viz. následující příklad), ovšem přesměrování lze realizovat i skriptovacími jazyky.

Příklad 3.18. Kontrola parametru *refresh* v hlavičce dokumentu

```
<sch:report test="translate(@http-equiv,'REFRESH','refresh') = 'refresh'">
  WCAG 1.0 Checkpoint 7.4 (Priorita 2) Uživatelé by neměli být ▶
nekontrolovatelně přesměrováváni.
</sch:report>
```

Jedním z požadavků deváté směrnice je i dostupnost ovládacích prvků nezávisle na použitém vstupním zařízení. Jednu z formalizací toho požadavku ukazuje následující příklad. Pokud je ošetřena událost vyvolaná počítačovou myší, měla by být ošetřena i odpovídající událost vyvolaná pomocí klávesnice.

Příklad 3.19. Pokud je na libovolném elementu ošetřena událost `onclick`, měla by být ošetřena i událost `onkeypress`.

```
<sch:rule context="*[@onclick]">
  <sch:assert test="@onkeypress">
    WCAG 1.0 Checkpoint 9.2 (Priorita 2) Pokud je na elementu <sch:name />
     ošetřena událost onclick měla by být ošetřena i událost onkeypress.
  </sch:assert>
</sch:rule>
```

Jedním z omezení desáté směrnice je i požadavek, aby nebyla používána tzv. „vyskakující“ okna *(pop-ups)*. Formalizace tohoto požadavku jde za hranice HTML a testuje přítomnost příslušných metod v obsahu elementu `<script>` pro dva široce rozšířené skriptovací jazyky. Tento přístup má ovšem několik omezení. Netestuje skripty obsažené v událostech nebo v externích souborech a podporuje pouze dva skriptovací jazyky (HTML je nezávislé na užitém skriptovacím jazyce).

Příklad 3.20. Kontrola volání funkcí pro otevření dodatečného okna v jazyce *javascript* a *vbscript*.

```
<sch:rule context="html:script">
  <sch:report test="@type = 'text/javascript' and ▶
```

```
contains(text(),'window.open(')">
    WCAG 1.0 Checkpoint 10.1 (Priorita 2) Není
    doporučeno používat "vyskakovací" okna.
  </sch:report>
  <sch:report test="@type = 'text/vbscript' and contains(text(),'MsgBox')">
    WCAG 1.0 Checkpoint 10.1 (Priorita 2) Není
    doporučeno používat "vyskakující" okna.
  </sch:report>
</sch:rule>
```

Formalizace jedenácté směrnice je zúžena na kontrolu všech potlačených elementů a atributů jazyka HTML. Testy jsou v zásadě triviální a kontrolují pouze výskyt těchto prvků. Kompletní seznam potlačených elementů a atributů je obsažen ve specifikaci HTML 4.01 viz. [HTML4].

Poslední směrnicí, kde se podařilo formalizovat některé z omezení, je dvanáctá směrnice. Dvanáctá směrnice požaduje, aby ovládací prvky využívaly explicitní popisky. V tomto případě by měl být element `<input>` obalen elementem `<label>`, jehož atribut `for` by se měl shodovat s příslušným atributem `id` tohoto ovládacího prvku. Tento požadavek je formalizován v následujícím příkladu.

Příklad 3.21. Explicitní popisky.

První test zjišťuje přítomnost rodičovského elementu `<label>` a příslušných atributů. Druhý test potom kontroluje shodnou hodnotu těchto atributů.

```
<sch:rule context="html:input">
  <sch:report test="@type = 'text' and (not(parent::html:label) or ▶
not(parent::html:label[@for]) or not(@id))">
    WCAG 1.0 Checkpoint 12.4 (Priorita 2) Ovládací prvky by měly mít ▶
explicitně asociovány popisky pomocí elementu label.
  </sch:report>
  <sch:report test="@type = 'text' and (string(parent::html:label/@for) ▶
!= string(@id))">
    WCAG 1.0 Checkpoint 12.4 (Priorita 2) Atribut for elementu label musí ▶
odpovídat atributu id příslušného elementu input.
    "<sch:value-of select="string(@id)"/>" != "<sch:value-of ▶
select="string(parent::html:label/@for)"/>"
  </sch:report>
</sch:rule>
```

3.6. Shrnutí

Dá se říci, že pomocí Relax NG a Schematronu lze formalizovat velkou část dodatečných omezení kladených na HTML dokumenty. Jde o ta omezení, která ze své podstaty nevyžadují lidský zásah a lidské posouzení. Exaktně definovaná a kvantifikovaná omezení, která nelze vyjádřit pomocí DTD, lze většinou dobře realizovat hlavně díky regu-

lárním výrazům a jazyku XPath. Velký podíl na expresivitě moderních validačních jazyků mají právě tyto dvě technologie. Zlepšování záběru automatické validace HTML dokumentů může vést k většímu dodržování standardů. Automatická validace sice nikdy nenahradí lidský faktor a znalost smyslu a záměru omezení definovaných ve standardech, ale může výrazně urychlit validační proces a částečně ulehčit roli lidského činitele v něm.

Formalizace omezení kladených na jazyk ISO-HTML je nad rámec této práce.

Kapitola 4

Implementace prototypové validační aplikace

4.1. Úvod

V této kapitole se budeme zabývat návrhem a implementací prototypové aplikace pro validaci HTML dokumentů oproti definici navržené v předešlé kapitole. Popsána bude užitá technologie, jednotlivé komponenty, jejich vzájemná interakce a ukázka užití prototypové aplikace v praxi.

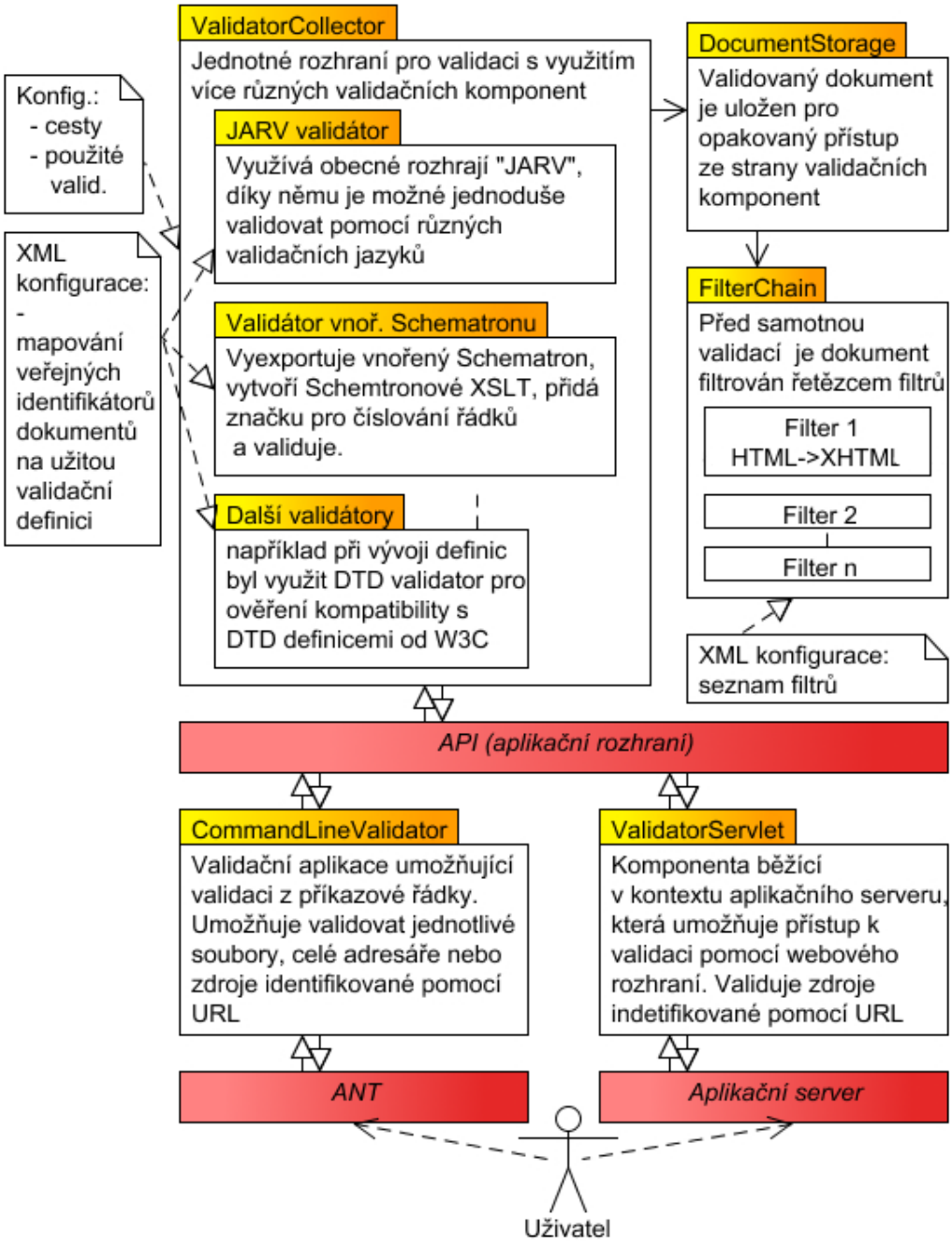
4.2. Architektura

Prototypová aplikace (v textu někdy označována také pouze jako aplikace) je složena z několika vzájemně spolupracujících komponent. Cílem návrhu byla jednoduchá použitelnost aplikačního rozhraní, dobrá rozšiřitelnost aplikace a co největší konfigurovatelnost prostřednictvím konfiguračních souborů. Architekturu aplikace znázorňuje obrázek 4.1 – „Architektura prototypové aplikace“.

Ústřední komponentou je `ValidatorCollector`. Ten nabízí jednotné a jednoduché validační rozhraní nezávisle na počtu užitých validačních komponent. Metoda pro zahájení validace je tak zavolána pouze jednou. `ValidatorCollector` se sám postará o správnou propagaci tohoto volání jednotlivým validačním komponentám. Navíc se stará o korektní agregaci všech chybových zpráv vzniklých při procesu validace.

Aplikace, které využívají rozhraní komponenty `ValidatorCollector`, poskytují validovaný dokument v podobě proudu dat *(stream)*. Tento proud dat je však v průběhu validace opakovaně využíván jednotlivými validačními komponentami. Pro opakovaný přístup je tedy nutné data získaná z datového proudu vhodným způsobem uskladnit. Tento úkol plní komponenta *DokumentStorage*. Tato komponenta se také stará o filtrování obsahu dokumentů ještě před samotnou validací. K tomuto účelu využívá posloupnost filtrů organizovaných do určitého řetězce. Předchozí filtr vždy poskytuje filtrovaná data filtru následujícímu. Tento řetězec je sestavován dynamicky na základě konfiguračního souboru, který určuje nejen užité filtry, ale i jejich pořadí v řetězci.

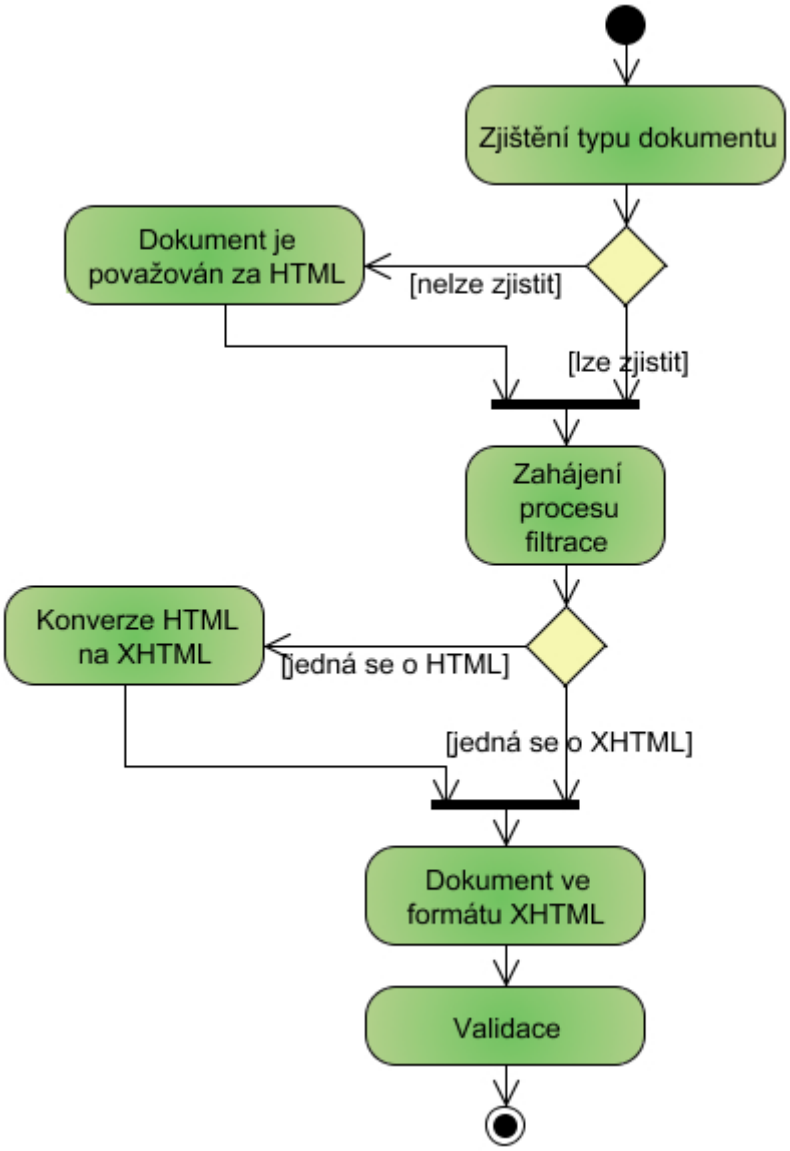
Příkladem využití řetězce filtrů je filtr, který je součástí prototypové aplikace a který dovoluje konverzi dokumentů psaných v HTML na formát XHTML. Tento filtr tedy umožňuje validovat i HTML dokumenty pomocí validačních jazyků určených pro validaci XML. Předpokladem validace takových dokumentů je samozřejmě



Obrázek 4.1. Architektura prototypové aplikace

úspěšná konverze. Obrázek 4.2 – „Konverze HTML dokumentů do formátu XHTML“ ukazuje proces, který takovouto konverzi zajišťuje. Cílem tohoto procesu je garantovat validovatelnost dokumentů pomocí zvolených validačních jazyků.

Po zahájení validace jsou postupně aktivovány jednotlivé validační komponenty a dochází k agregaci jejich výstupů. Validační komponenty využívají specifický druh konfigurace, který mapuje veřejný identifikátor dokumentů *(publicId)* na příslušnou definici, která má být pro daný typ dokumentu užita k validaci. Takovéto mapování je dále strukturováno do skupin podle validačního parametru, který je zadáván uživatelem. Tento parametr v zásadě dává uživateli prostor zvolit si režim validace. Typicky



Obrázek 4.2. Konverze HTML dokumentů do formátu XHTML

je možné vybrat standard, proti kterému je vstupní dokument validován. Následující příklad ukazuje takovouto konfiguraci.

Příklad 4.1. Mapování veřejného identifikátoru dokumentů na validační definice.

Následující konfiguraci využívají validátory v prototypové aplikaci. Vidíme, že jsou definovány dva validační parametry. První parametr `(xhtml)` umožňuje validaci podle doporučení W3C pro jazyk HTML. Druhý parametr `(xhtml+wcag)` navíc do validačního procesu zapojuje i pravidla pro dostupnost obsahu WCAG. Tyto parametry jsou specifikovány uživatelem při požadavku na validaci. Atribut `desc` může být zobrazován uživatelským rozhraním a může tedy sloužit k lepší orientaci v nabízených možnostech.

Hlavním cílem této konfigurace je informovat validační komponenty o tom, jakou validační definici mají použít pro jednotlivé typy HTML dokumentů.

```
<option name="xhtml" desc="W3C XHTML 1.0 / HTML 4.01">
  <entry default="true" ▶
schemaPath="/conf/schema/rng/xhtml-transitional.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0 Strict//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0 Transitional//EN" ▶
schemaPath="/conf/schema/rng/xhtml-transitional.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0 Frameset//EN" ▶
schemaPath="/conf/schema/rng/xhtml-frameset.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01 Strict//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01 Transitional//EN" ▶
schemaPath="/conf/schema/rng/xhtml-transitional.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01 Frameset//EN" ▶
schemaPath="/conf/schema/rng/xhtml-frameset.rng" />
</option>
<option name="xhtml+wcag" desc="WAI WCAG 1.0 and W3C XHTML 1.0 / HTML ▶
4.01">
  <entry default="true" ▶
schemaPath="/conf/schema/rng/xhtml-transitional-wcag.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict-wcag.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0 Strict//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict-wcag.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0 Transitional//EN" ▶
schemaPath="/conf/schema/rng/xhtml-transitional-wcag.rng" />
  <entry publicId="-//W3C//DTD XHTML 1.0 Frameset//EN" ▶
schemaPath="/conf/schema/rng/xhtml-frameset-wcag.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict-wcag.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01 Strict//EN" ▶
schemaPath="/conf/schema/rng/xhtml-strict-wcag.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01 Transitional//EN" ▶
schemaPath="/conf/schema/rng/xhtml-transitional-wcag.rng" />
  <entry publicId="-//W3C//DTD HTML 4.01 Frameset//EN" ▶
schemaPath="/conf/schema/rng/xhtml-frameset-wcag.rng" />
</option>
```

4.3. Validační komponenty

První validační komponentou je tzv. JARV *(Java Application Programming Interface for RELAX Verifiers)* validátor. JARV je validační rozhranní, které je nezávislé na použité implementaci validátoru. Velkou výhodou využití takového rozhraní je možnost validovat s pomocí téměř libovolných validačních jazyků bez nutnosti změny kódu apli-

kace. Stačí změnit konfiguraci v některém z konfiguračních souborů. Toto rozhraní tedy dává validační komponentě možnost jednoduchého rozšíření. Ve stávající aplikaci je ovšem využívána pouze validace oproti definicím zapsaným v Relax NG. V případě potřeby je ovšem možné stejnou validační komponentu jednoduše použít pro validaci definic zapsaných v XML Schema nebo v některém jiném podporovaném validačním jazyce.

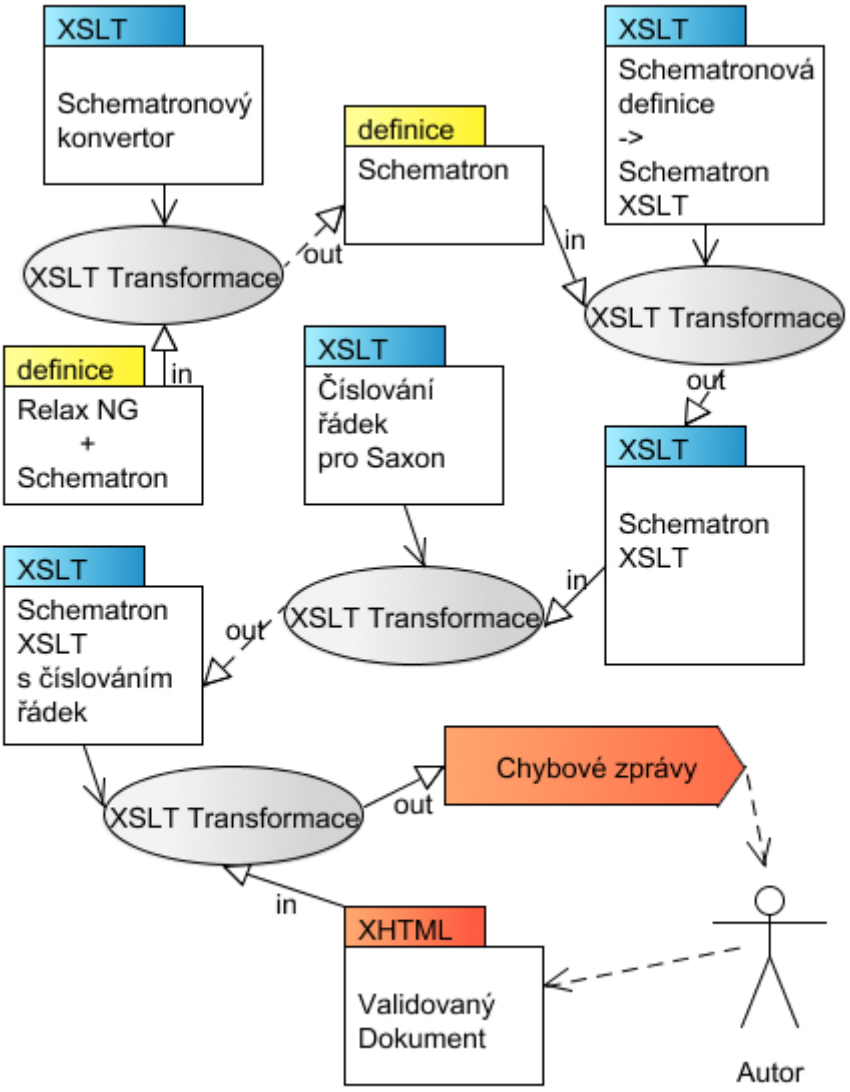
Další validační komponentou je komponenta umožňující validaci vnořeného Schematronu. Návrh validační komponenty je opět velmi otevřený. Pouhou změnou konfiguračních souborů a napsáním příslušných transformačních stylů je možné uskutečnit validaci Schematronu vnořeného v některém jiném validačním jazyce, než je použité Relax NG. Validace je v tomto případě v podstatě sérií čtyř po sobě jdoucích XSLT transformací. Průběh těchto transformací znázorňuje obrázek 4.3 – „Transformační proces při validaci vnořeného Schematronu“.

První transformační proces transformuje vstup, což je v našem případě definice v jazyce Relax NG s vnořenými Schematronovými testy, na čistý Schematron, který je získáván ze všech zahrnutých modulů definice. Schematronová definice je dále transformována na Schematronový transformační styl. Ten je již přímo možné použít k transformaci validovaného dokumentu, jejíž výsledkem by byla příslušná chybová hlášení. Z praktických důvodů je však lepší zahrnout do takových hlášení ještě čísla řádek, na kterých došlo ve validovaném dokumentu k chybám. Tato informace výrazně zvyšuje použitelnost validačního výstupu a urychluje odstraňování chyb. Získání čísel řádku zdrojového dokumentu v průběhu transformace ovšem nepatří mezi standardní funkce transformačních nástrojů. Pro implementaci této validační komponenty byl vybrán transformační nástroj Saxon, který obsahuje proprietární rozšíření pro zjištění čísel řádků. Další transformační proces tedy přidává do Schematronového transformačního stylu speciální značku, která je v průběhu transformace rozeznána Saxonem a nahrazena ve výstupu příslušným číslem řádku.

Do posledního transformačního procesu vstupuje konečně samotný validovaný dokument. Ten je transformován pomocí Schematronového stylu, který jsme získali předešlými kroky. Výsledkem jsou pak konkrétní chybové zprávy opatřené i číslem řádku, na kterém byla chyba objevena. Tyto zprávy je možné spolu se zprávami z ostatních validačních komponent nakonec zobrazit koncovým uživatelům aplikace.

Všechny validační komponenty jsou potomky abstraktní validační třídy. Tato třída v sobě zahrnuje společnou funkcionalitu validačních komponent. Tato funkcionalita se týká i správy validačních definic. Je zřejmé, že před samotnou validací je třeba validační definice náležitě připravit. Tato příprava je relativně náročná na zdroje. Například v případě validátoru vnořeného Schematronu obnáší tři transformační procesy. Proto je dobré připravené definice uchovávat k opětovnému využití. Tuto službu zajišťuje právě abstraktní validační třída. Její potomci implementují pouze specifické způsoby přípravy definice a samotný validační proces, který je také specifický pro každou validační komponentu.

Specifický je i způsob mapování veřejných identifikátorů na definice. Každá validační komponenta tak má možnost využívat vlastní mechanizmy k tomuto účelu. Lze například využít jiný formát mapovacího konfiguračního souboru a nebo třeba používat vždy stejnou validační definici bez ohledu na veřejný identifikátor dokumentu.



Obrázek 4.3. Transformační proces při validaci vnořeného Schematronu

4.4. Uživatelské rozhraní

Prototypová aplikace nabízí uživateli dva základní přístupy k validační funkcionalitě. První z nich je validační aplikace ovládaná z příkazové řádky. Ta umožňuje validovat jednotlivé soubory, celé adresáře nebo zdroje určené pomocí URL. Příkladem využití takové aplikace je validace všech testových úloh, které jsou součástí prototypové aplikace. Následující příklad ukazuje nápovědu validační aplikace ovládané z příkazové řádky. Lze si tak udělat obrázek o možnostech jejího použití.

Příklad 4.2. Validace z příkazové řádky.

```
CommandLineValidator - použití:
Povinné parametry:
  -o <validační parametr>  validační parametr
  -f | -r | -w             alespoň jedna z následujících
                           voleb musí být specifikována:
                           kořenový adresář, jednotlivý soubor
                           nebo zdroj identifikovaný pomocí URL
Nepovinné parametry:
  -f <jednotlivý soubor>   validuje jednotlivý soubor, více souborů
                           může být specifikováno opakováním tohoto
                           parametru
  -w <URL>                 zdroj identifikovaný pomocí URL, více zdrojů
                           lze specifikovat opakováním tohoto parametru
  -r <kořenový adresář>    absolutní cesta ke kořenovému adresáři
  -d <list podadresářů>    tyto adresáře (relativní ke kořenovému)
                           jsou zahrnuty resp. vyjmuty z validace
  -i                       adresáře definované parametrem -d jsou
                           zahrnuty do validace
  -e (implicitně platí)    adresáře definované parametrem -d jsou
                           vyjmuty z validace
  -help                    vypíše tuto nápovědu
  -s                       tichý výstup, vypíše pouze testové úlohy,
                           které mají být validní resp. chybné a ve
                           skutečnosti platí opak
```

Pro snadnější ovládání obsahuje aplikace několik předdefinovaných konfigurací. Dvě z nich slouží jako automatizovaná podpora testové fáze vývojového cyklu viz. sekce 3.3 – „Metodika zápisu formalizací – vývojový cyklus“. První z nich postupně otestuje všechny testové úlohy v adresáři `new`, kde jsou vyvíjeny nové testové úlohy, které zachycují nějaký nově implementovaný jev. Teprve potom co považujeme implementaci tohoto jevu za dostatečně stabilní a všechny testové úlohy jsou úspěšné, můžeme přistoupit k druhé předdefinované konfiguraci. Ta je nastavena tak, aby validovala všechny testové úlohy, které jsou relevantní pro danou definici. Je nutné zajistit, aby nově naimplementovaný prvek nenarušil některá již implementovaná omezení. Pokud zjistíme nějaké nové nesrovnalosti, je třeba příslušným způsobem opravit definici. Celý proces opakujeme tak dlouho, dokud nejsou všechny testové úlohy v pořádku. Poté je možné zařadit nové testové úlohy do příslušné struktury mezi ostatní. Tento proces umožňuje plynulou implementaci a pomáhá udržovat definici konzistentní v celém průběhu vývojového cyklu.

Další možností přístupu k validační funkcionalitě je webové rozhraní. Zatímco příkazová řádka slouží spíše k účelům testování a správy validačních definic, webové rozhraní je určeno k použití skutečným autorům HTML dokumentů. Ti mají možnost prostřednictvím webového formuláře zadat URL svého dokumentu a příslušný vali-

dační parametr. Výstupem je potom seznam chybových zpráv. Rozhraní se tedy používá velice jednoduše a intuitivně.

Obrázek 4.4 – „Využití validační aplikace prostřednictvím prohlížeče Firefox 1.0.2“ ukazuje validační výstup pro hlavní stránku webového vyhledávače Google. Z výstupu je patrné, že nebyl rozpoznán typ dokumentu (ten není ve zdrojové stránce vůbec specifikován), a proto byl použit implicitní typ. Zdroj byl tedy konvertován na XHTML a posléze validován. Zbytek viditelného výstupu jsou validační chyby.

Na obrázku 4.5 – „Využití validační aplikace prostřednictvím prohlížeče Internet Explorer 6.0“ vidíme validační výstup pro hlavní stránku magazínu Interval.cz. Uživatelem byla zvolena také validace směrnic WCAG. Stránky jsou psány v XHTML a konverze tedy v tomto případě nebyla nutná.

Na obrázku 4.6 – „Stránky abclinux.cz validované v textovém prohlížeči Lynx 2.8.5“ vidíme validační výstup v textovém prohlížeči Lynx. I přestože se jedná o dokument v jazyce HTML, byl typ tohoto dokumentu správně rozeznán a byla provedena konverze na XHTML.

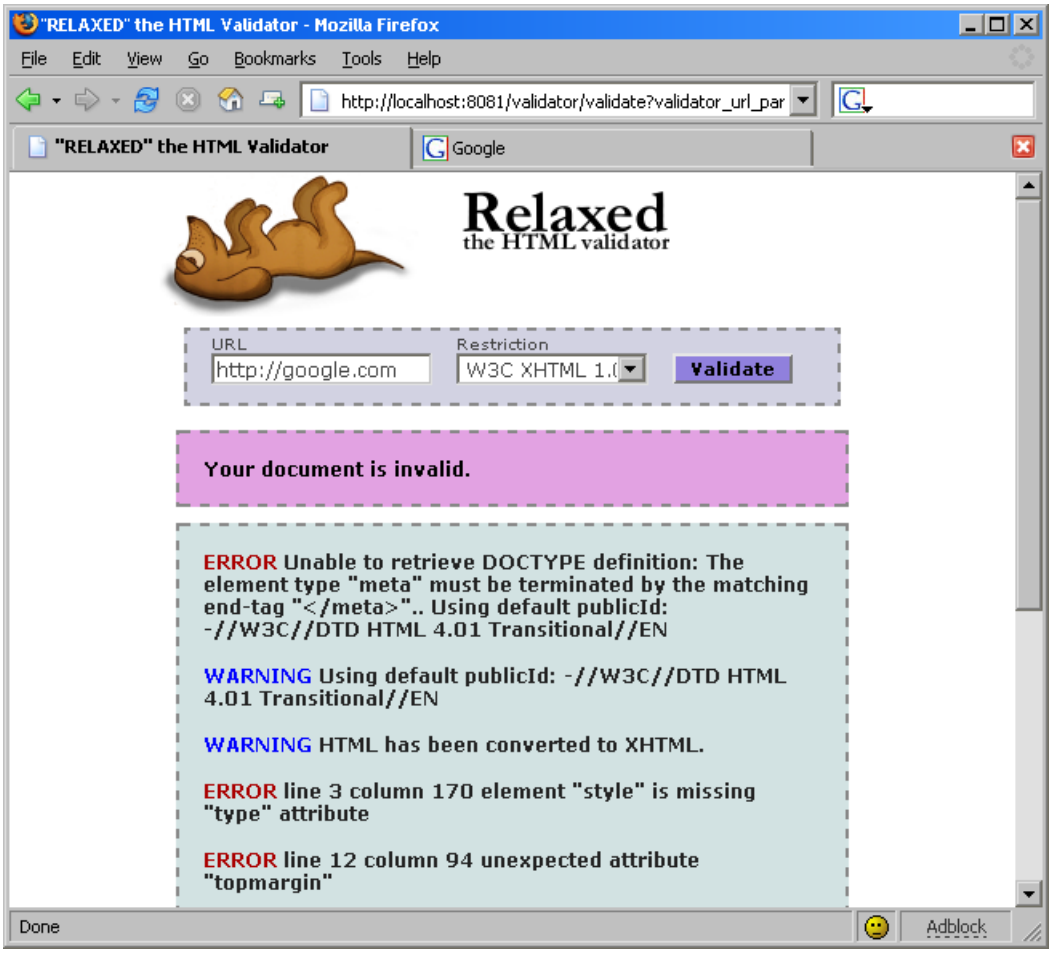
Na závěr je připravena ukázka výstupu validace samotné hlavní stránky prototypové validační aplikace viz. obrázek 4.7 – „Bezchybný výstup“. Jsou validovány i směrnice WCAG.

4.5. Technologie

Prototypová aplikace je napsána s využitím programovacího jazyka Java 1.4. Pro běh aplikace je nutná celá řada knihoven, které jsou součástí distribuce této aplikace. Jedná se například o výše popsané validační rozhraní JARV 1.4. Toto rozhraní dále používá validační nástroj Sun Multi-Schema Validator (MSV) 1.2, který podporuje validaci s využitím celé řady validačních jazyků (Relax NG a XML Schema nevyjímaje). XSLT transformace zajišťuje Saxon 6.5.3.

O konverzi HTML na XHTML se stará knihovna Tag Soupe. Jedná se o relativně malý a velice mladý projekt, jehož cílem je zpřístupnit HTML dokumenty XML nástrojům. Na rozdíl od jiných podobných knihoven se vůbec nezabývá zkrášlováním HTML zdroje. Jeho jediným cílem je zajistit výstup ve správně formovaném XML/XHTML, které nebude problematické pro další zpracování. Prioritou konverzního procesu je vyprodukovat nějaký použitelný výstup nezávisle na chybách, ke kterým mohlo v průběhu konverze dojít. Tento přístup je ideální pro využití v prototypové aplikaci, protože vždy dovoluje doručit uživatelům informace o chybách v jejich dokumentech. Může se ovšem stát, že některé zásadní chyby nebudou odhaleny. Protože by jejich přítomnost znemožňovala další zpracování, jsou tyto chyby knihovnou Tag Soupe opraveny, a neobjeví se tak v chybovém výstupu.

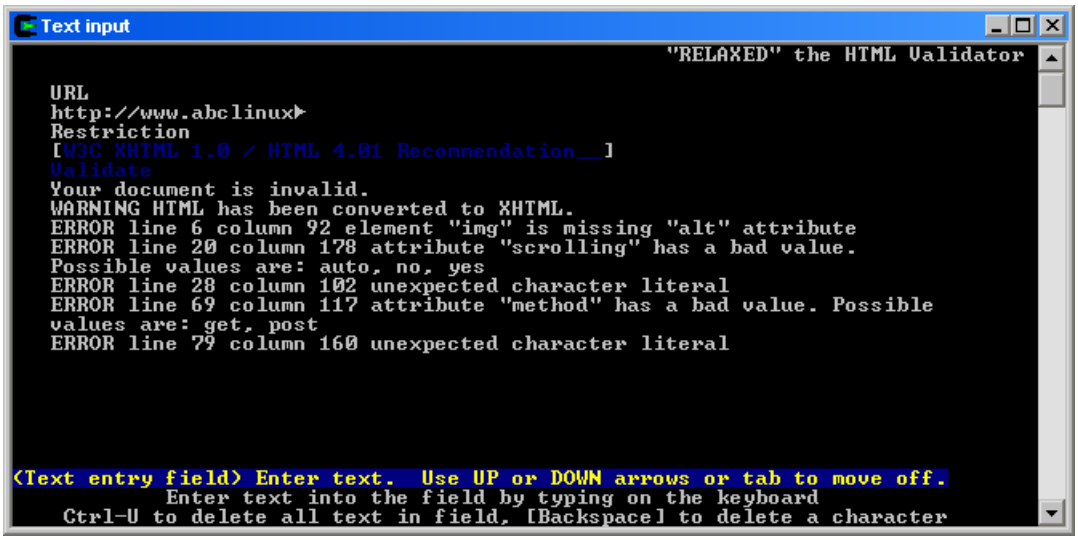
Aplikace dále využívá distribuční nástroj ANT od Apache Foundation. Ten umožňuje automatizovaný výkon úloh, které jsou nezbytné pro provoz aplikace. Definice těchto úloh je součástí aplikace a je zapsána v konfiguračních souborech psaných v XML. K základním úlohám patří kompilace zdrojových kódů, generování dokumentace a instalace webového rozhraní na příslušný aplikační server (aplikace byla testována na serveru Tomcat 5.0.14). Další úlohy potom umožňují automatizované provádění testových úloh s různými parametry a nebo validaci jednotlivých HTML dokumentů z příkazové řádky.



Obrázek 4.4. Využití validační aplikace prostřednictvím prohlížeče Firefox 1.0.2

Další informace o implementaci prototypové aplikace lze nalézt v dokumentačním adresáři `doc` v adresářové struktuře přiložené validační aplikace. Tento adresář obsahuje (kromě textu této práce) stručnou dokumentaci jednotlivých javovských tříd. Tato dokumentace je vygenerována pomocí standardního dokumentačního nástroje javadoc, který je součástí vývojového balíku programovacího jazyka Java.

Obrázek 4.5. Využití validační aplikace prostřednictvím prohlížeče Internet Explorer 6.0



Obrázek 4.6. Stránky abclinux.cz validované v textovém prohlížeči Lynx 2.8.5

Obrázek 4.7. Bezchybný výstup

Kapitola 5

Závěr

Tato práce jasně prokázala, že pomocí moderních validačních jazyků je skutečně možné validovat některá dodatečná omezení vycházející ze standardů HTML. Dále ukázala, jaký přínos mají tyto technologie pro oblast validace oproti tradičním přístupům.

Práce může sloužit nejen jako přehled různých standardů jazyka HTML a různých druhů omezení, které tyto standardy definují, ale jedná se i o určité srovnání možností jednotlivých validačních jazyků, které může případným čtenářům pomoci s výběrem vhodného validačního řešení pro jejich konkrétní potřeby. Výstupem práce je navíc validační definice jazyka HTML 4.01 / XHTML 1.0 a směrnic WCAG 1.0 spolu s plně funkčním testovacím a vývojovým prostředím, které lze použít pro případné další rozšiřování definice a implementaci nových standardů. HTML dokumenty lze validovat i pomocí webového rozhraní. Zde se otevírají další možnosti pro autory těchto dokumentů. Ti mohou využívat výsledky této práce jako veřejně dostupnou validační službu, která může pomoci zajistit lepší soulad jejich dokumentů se standardy. Je ovšem nutné mít na paměti, že tato validační aplikace nikdy nahradí podrobnou znalost smyslu a účelu standardů HTML. Může však usnadnit validační proces a automatizovaně kontrolovat co nejvíce formalizovatelných omezení.

Vlastním přínosem autora je hlavně určitá sumarizace omezení vybraných z jednotlivých standardů HTML, dále pak analýza a výběr vhodných validačních řešení, implementace dodatečných omezení pomocí tohoto řešení a samozřejmě návrh a implementace prototypové validační aplikace.

Tato práce nechává široký prostor pro případné práce návazné. Existuje celá řada standardů, které je možné formalizovat a umožnit v rozšířené míře i jejich validaci. Validační definice je dobře připravena pro plnou implementaci modularizace XHTML a mutace XHTML Basic. Přínosem by byla i formalizace omezení vyplývajících z ISO-HTML, kde by bylo možné podchytit některá omezení, která nejsou vyjádřena v DTD tohoto jazyka. Také by mohl být usnadněn publikační proces, který vyžaduje ještě dodatečnou validaci v průběhu zpracovávání dokumentů.

Validační aplikace je psána se záměrem jednoduché rozšiřitelnosti. V případě potřeby lze implementovat další validační komponenty, například komponentu validující čistý Schematron (nikoli vnořený). Tato komponenta může urychlit validaci v případě, že budou některá omezení formalizována pouze ve Schematronu.

Zajímavým rozšířením by mohlo být i využití Schematronu s expresivnější verzí jazyka XPath 2.0. To by umožnilo podchytit i některá omezení, která nebylo možné

v této práci formalizovat. V tomto kontextu může být zajímavé i využití nové verze Schematronu, která má být již brzy vydána.

Drobným vylepšením webové validační aplikace může být možnost poslat validovaný dokument kompletně k validaci pomocí HTTP protokolu v případě, že ho uživatel nemá vystaven v síti Internet, a není tedy přístupný pomocí URL.

V případě implementace modularizace XHTML bude nutné rozšířit systém výběru validačních možností pomocí validačního parametru tak, aby bylo možné validovat libovolné moduly poskládané v různých kombinacích.

Seznam použité literatury

[HTML4] Ragget, D., Le Hors, A., Jacobs, I.: HTML 4.01 Specification. W3C, 1999. Dostupný z WWW: http://www.w3.org/TR/1999/REC-html401-19991224/

[ISO15445] ISO/IEC: HyperText Markup Language (HTML), ISO/IEC 15445:2000(E). ISO/IEC, 2000. Dostupný z WWW: https://www.cs.tcd.ie/15445/15445.HTML

[ISO15445UG] ISO/IEC: User's Guide to ISO/IEC 15445:2000 HyperText Markup Language (HTML). ISO/IEC, 2000. Dostupný z WWW: https://www.cs.tcd.ie/15445/UG.HTML

[XHTML1] XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). W3C, 2002. Dostupný z WWW: http://www.w3.org/TR/2002/REC-xhtml1-20020801/

[RFC2119] Bradner, S.: Key words for use in RFCs to Indicate Requirement Levels. Network Working Group, 1997. Dostupný z WWW: http://www.ietf.org/rfc/rfc2119.txt

[RFC3066] Alvestrand, H.: Tags for the Identification of Languages. Network Working Group, 2001. Dostupný z WWW: http://www.ietf.org/rfc/rfc3066.txt

[RFC2396] Berners-Lee, T., Fielding, R., Irvine, U., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax. Network Working Group, 1998. Dostupný z WWW: http://www.ietf.org/rfc/rfc2396.txt

[WCAG1] Chisholm, W., Vanderheiden, G., Jacobs, I.: Web Content Accessibility Guidelines 1.0. W3C WAI, 1999. Dostupný z WWW: http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/

[SCHTR] Jelliffe, R.: The Schematron Assertion Language 1.5. Academia Sinica Computing Centre, 2002. Dostupný z WWW: http://xml.ascc.net/resource/schematron/Schematron2000.html

[DSD] Møller, A.: Document Structure Description 2.0. BRICS, 2002. Dostupný z WWW: http://www.brics.dk/DSD/dsd2.html

[XMLSCH-ST] Thompson, H., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures Second Edition. W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[XMLSCH-DT] Biron, P., Malhotra, A.: XML Schema Part 2: Datatypes Second Edition. W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[XHTML-XMLSCH] Ishikawa, M.: XHTML™ 1.0 in XML Schema. W3C, 2002. Dostupný z WWW: http://www.w3.org/TR/2002/NOTE-xhtml1-schema-20020902/

[RNG] Clark, J., Murata, M.: RELAX NG Specification. OASIS Committee, 2001. Dostupný z WWW: http://www.relaxng.org/spec-20011203.html

[XMLDTD] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0 (Third Edition). W3C, 2004. Dostupný z WWW: http://www.w3.org/TR/2004/REC-xml-20040204/

[LEE-CHU] Lee, D., Chu, W.: Comparative Analysis of Six XML Schema Languages. Department of Computer Science University of California, 2000. Dostupný z WWW: http://www.cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.html

[VLIST] van der Vlist, E.: Using W3C XML Schema. XML.com, 2001. Dostupný z WWW: http://www.xml.com/pub/a/2000/11/29/schemas/part1.html?page=1

[XHTMLMOD] Altheim, M., McCarron, S., Boumphrey, F., Dooley, S., Schnitzenbaumer, S., Wugofski, T.: Modularization of XHTML™. W3C, 2001. Dostupný z WWW: http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/

[XHTML11] Altheim, M., McCarron, S.: XHTML™ 1.1 – Module-based XHTML. W3C, 2001. Dostupný z WWW: http://www.w3.org/TR/2001/REC-xhtml11-20010531/

[XHTMLMOD-RNG] Clark, J.: Modularization of XHTML in RELAX NG. Thai Open Source Software Center Ltd, 2003. Dostupný z WWW: http://www.thaiopensource.com/relaxng/xhtml/

[NRL] Clark, J.: Namespace Routing Language (NRL). Thai Open Source Software Center Ltd, 2003. Dostupný z WWW: http://www.thaiopensource.com/relaxng/nrl.html

Slovníček pojmů

A

Atribut
: *Attribute* – Atributem se v této práci většinou myslí atribut ve smyslu SGML. Jde v podstatě o pojmenovanou hodnotu asociovanou s elementem. Atributy se zapisují v počáteční značce elementu za jeho jménem ve formátu jméno=hodnota. V XML je nutné vždy hodnoty uzavírat do uvozovek.
Viz též Element, Značka, XML, SGML.

B

Booleovský atribut
: Booleovským atributem je myšlen atribut, který může nabývat pouze dvou hodnot. Můžeme je nazývat třeba „pravda“ a „lež“. V SGML/XML rozhoduje o této hodnotě přítomnost či nepřítomnost takového atributu v elementu.
Viz též Atribut, Element, Značka, XML, SGML.

C

CSS
: *Cascading Style Sheets* – Kaskádové styly jsou jedním ze stylových jazyků. Jedná se o doporučení W3C. Stylové definice připojené k dokumentům popisují, jakým způsobem mají být jeho jednotlivé elementy zobrazovány. K dokumentům může být připojeno více stylových definic s různými prioritami. Vytváří tedy určitou kaskádu. Odtud také název jazyka.
Viz též Stylový jazyk, Stylová definice.

D

Datový typ
: *Data type* – Jedná se o jméno, nebo označení určité množiny hodnot spolu s operacemi, které je možné nad těmito hodnotami vykonávat. Datový typ definuje určitá omezení, které musí splňovat hodnoty příslušející tomuto datovému typu.

DTD
: *Document Type Definition* – DTD je jazyk umožňující formální definici elementů, struktury a pravidel značkování pro značkovací jazyky odvozené ze SGML. Proti takovým definicím je pak možné validovat různé dokumenty, a zjišťovat

tak jejich soulad se značkovacím jazykem, ve kterém jsou napsány.
Viz též XML, XML DTD, SGML.

E

Element — Element je základním stavebním kamenem značkovacích jazyků. Je nositelem atributů a obsahu. Kromě textových informací může element obsahovat další elementy (subelementy). V dokumentech je element a jeho obsah většinou vymezen počáteční a koncovou značkou.
Viz též Atribut, Značka, SGML, XML.

H

HTML — *Hyper Text Markup Language* – Jedná se o jazyk, ve kterém je zapisován obsah jednotlivých WWW-stránek. Je založen na principu značkování částí textu pomocí značek (*markup*), které přiřazují dodatečný význam textu. Například to, že určitá část textu představuje nadpis. Jiné značky zase specifikují, že na určité místo má být vložen obrázek, že určitá část textu představuje hypertextový odkaz apod. Podle těchto značek se pak při zobrazování stránek v HTML prohlížeči určuje jejich grafická, nebo jiná reprezentace. Jazyk je aplikací SGML, nebo XML. V případě XML se také často označuje jako XHTML.
Viz též WWW, Hypertext, XML, SGML.

HTTP — *Hyper Text Transfer Protocol* – HTTP je komunikační protokol určený pro komunikaci s webovými servery. Typickou funkcí tohoto protokolu je zajistit klientské aplikaci (většinou se jedná o prohlížeč) spojení s webovým serverem a vyžádat si od něj HTML dokumenty (stránky), které jsou následně přeneseny do klientské aplikace.
Viz též Webový server, WWW, Hypertext.

Hypertext — Text členěný na menší celky (označované jako stránky), které jsou mezi sebou provázány odkazy.

I

IANA — *Internet Assigned Numbers Authority* – Úřad pro správu přidělených „čísel“ sítě Internet. Protokoly Internetu mají různé parametry, např. čísla portů, protokolů, různé kódy apod. IANA je ústřední registr těchto parametrů. Přiřazené hodnoty publikuje ve speciálních dokumentech.

J

JARV — *Java Application Programming Interface for RELAX Verifiers* – Jedná se o aplikační rozhraní pro validaci, které je nezávislé na užité validační implementaci a užitém validačním jazyce. Viz též Validační jazyk, Validační nástroj.

Java applet — Aplikace napsaná v programovacím jazyce Java, která může být spuštěna přímo v rámci HTML dokumentu. Takováto aplikace je nejprve stažena prohlížečem (pokud podporuje tuto technologii) a poté je spuštěna na straně klienta.

Jmenný prostor — *Namespace* – Jmenné prostory označují určitou skupinu jmen, které se dají pomocí jmenných prostorů adresovat a v jejichž kontextu jsou unikátní. Určité jméno je tak jednoznačně identifikováno jmenným prostorem, ke kterému náleží. Stejná jména v různých jmenných prostorech mají také různý význam. Jmenné prostory v XML umožňují využít více různých XML jazyků v rámci jednoho dokumentu. Různé jmenné prostory jsou v XML identifikovány pomocí URI.
Viz též XML, URI.

M

MathML — *Mathematical Markup Language* – Značkovací jazyk určený k vyjádření matematických rovnic. Jedná se o aplikaci XML.
Viz též XML.

O

OOP — *Object Oriented Programming* – Programovací technika orientovaná na objekt, což je datová struktura zapouzdřená s příslušnými metodami, které operují nad těmito datovými strukturami. Objektové programování stojí na několika základních metodách, které umožňují vysokou míru modularity a znovupoužitelnosti vytvářených struktur. Jedná se hlavně o abstrakci, zapouzdření, polymorfizmus a dědičnost.
Viz též Přepis.

P

Parser — Syntaktický analyzátor – Jedná se o aplikaci, která rozeznává strukturu a data určitého jazyka. Vstupní dokument je rozdělen na jednotlivé části. Ty jsou dále poskytnuty jiným komponentám k dalšímu zpracování.
Viz též XML, SGML.

Potlačené — *depricated* – Jako potlačené se označují ty elementy nebo atributy značkovacích jazyků, které není doporučené nadále používat a které zůstaly v definici pouze z důvodu zpětné kompatibility.
Viz též HTML.

Prohlížeč — *browser* – Jako prohlížeč (HTML prohlížeč) v této práci označujeme aplikaci, která je schopna nějakým způsobem zprostředkovat HTML dokumenty jejich uživatelům. Takové zprostředkování není závislé pouze na vizuálním zobrazování, ale dokument může být například i čten a nebo zprostředkováván v Braillově písmu.
Viz též HTML.

Přepis — *Overwriting* – Jedna z metod objektového programování, která umožňuje přepsat nějaké chování nebo datové struktury zděděné potomkem (objektem) od předka.
Viz též OOP.

R

Regulární výrazy — *Regular expressions* – Jedná se o jazyk umožňující popsat určité vzory textový dat. Pomocí regulárních výrazů lze posléze rozhodnout, zda určitý textový řetězec odpovídá danému vzoru (nebo nikoli).
Viz též Validační jazyk.

S

SGML — *Standard Generalized Markup Language* - SGML není sám o sobě značkovacím jazykem. Jedná se o ISO standard, který popisuje způsob definování značkovacích jazyků. Jde v podstatě o jakýsi metajazyk. Příkladem jazyků postavených na SGML je například HTML nebo XML.
Viz též HTML, XML.

Skriptovací jazyk — *Scripting language* – V této práci se zabýváme skriptovacími jazyky, které jsou použitelné v rámci HTML dokumentů. Jedná se o zjednodušené programovací jazyky, které mají přístup k objektovému modelu HTML dokumentů a umožňují nad ním dynamicky vykonávat určité manipulace. Skripty v příslušném jazyce jsou v podstatě sérií příkazů, které mohou být sdruženy do funkcí. Takové příkazy (nebo funkce) mohou být volány v rámci ošetření různých událostí, ke kterým může v HTML dokumentech docházet (stisk klávesy, načtení dokumentu, ...).
Viz též Skriptování.

Skriptování — *Scripting* – Skriptováním se v této práci myslí vytváření dodatečné funkcionality HTML dokumentů pomocí sérií příkazů zapsaných v skriptovacím jazyce.

	Viz též Skriptovací jazyk.
Stylová definice	*Stylesheet* – Jedná se o popis vizuální reprezentace určitého dokumentu v příslušném stylovém jazyce. Viz též Stylový jazyk, CSS.
Stylový jazyk	*Stylesheet language* – Jedná se o jazyk popisující vizuální stránku (tzv. styl) elementů značkovacích jazyků. Viz též Stylová definice, CSS.
Subelement	V hierarchické struktuře prvků značkovacích jazyků je jako subelement označen takový element, který je uzavřen bezprostředně uvnitř jiného elementu. Viz též Element.
SVG	*Scalable Vector Graphics* – SVG je jazyk, který popisuje dvoudimenzionální vektorovou grafiku pomocí XML. Viz též XML.

T

Transformační nástroj	Transformačním nástrojem v této práci rozumíme takový nástroj, který transformuje XML dokumenty na jiné dokumenty. Tento transformační proces je pak popsán pomocí jazyka XSLT. Viz též XML, XSLT.

U

URI	*Uniform Resource Identifier* – URI je v zásadě určitý formátovaný řetězec, který identifikuje nějaký zdroj (typicky zdroj v síti internet). Viz též URL.
URL	*Uniform Resource Locator* – URL je specifickým případem URI. Jedná se o identifikátor zdrojů v síti Internet. Typicky určuje metodu, pomocí které je daný zdroj dostupný, jméno počítače, kde lze zdroj nalézt, a cestu ke zdroji v rámci tohoto počítače. Viz též URI.

V

Validační jazyk	*Schema language* – V rámci této práce se jako validační označuje jazyk sloužící pro definici elementů, struktury a pravidel značkování ve značkovacích jazycích. Viz též XML DTD, DTD.
Validační nástroj	V této práci se za validační nástroj považuje programové vybavení, které ověřuje soulad nějakého značkovaného dokumentu s definicí značkovacího jazyka v příslušném validačním jazyce. Vstupem takového nástroje je instance validovaného dokumentu a příslušná definice. Výstupem

je pak rozhodnutí o validitě, které může být doplněno o seznam a popis nalezených chyb.
Viz též Validační jazyk.

Veřejný identifikátor — *Public identifier* – Veřejný identifikátor je součástí deklarace typu dokumentu v SGML a XML. Typem dokumentu se v zásadě myslí označení značkovacího jazyka, ve kterém byl příslušný dokument psán. Veřejný identifikátor má předepsanou syntaxi a je unikátní vzhledem k užité DTD definici značkovacího jazyka.
Viz též DTD, XML, SGML.

W

Webový server — *Web server* – Webový server můžeme chápat jako proces, který běží na nějakém počítači, jehož cílem je zprostředkovávat prostřednictvím HTTP protokolu určitý obsah (typicky HTML dokumenty) klientům na základě jejich požadavku.
Viz též WWW, HTML, HTTP.

Řádně formovaný — *Well-formed* – Tento termín se užívá pro dokumenty, které splňují základní syntaxi jazyka XML.
Viz též XML.

WWW — *World Wide Web* – Služba Internetu určená pro zpřístupnění různých druhů informací. Funguje na bázi architektury klient/server. Pro její fungování jsou zapotřebí WWW-servery a dále klientské programy (prohlížeče), prostřednictvím kterých uživatelé s touto službou pracují.

X

XML — *eXtensible Markup Language* – XML je v zásadě zjednodušeným dialektem jazyka SGML. Omezené možnosti a přísnější syntaxe jej dělají jednoduše použitelným a lépe zpracovatelným při zachování dobrých vyjadřovacích možností.
Viz též XML DTD, SGML.

XML DTD — XML DTD je užíváno pro popis elementů, struktury a značkovacích pravidel jazyků odvozených od XML
Viz též DTD, XML.

XPATH — *XML Path Language* – Jazyk XPATH byl navržen pro adresování určitých částí XML dokumentů. Umožňuje adresovat jednotlivé elementy, atributy nebo jejich skupiny v rámci struktury dokumentu a obsahuje funkce pro manipulaci s jejich hodnotami. XPath je využíván například v transformačních procesech popsaných v XSLT.
Viz též XML, XSLT.

XSLT

eXtensible Stylesheet Language Transformations – XSLT je jazyk popisující transformační procesy pro transformaci XML dokumentů na dokumenty jiné.
Viz též XML, XPATH.

Z

Značka

Značky slouží k vymezení určitých částí textu, kterým přiřazují dodatečné významy. Zpravidla mluvíme o počáteční a koncové značce, které vymezují text mezi nimi. Značky jsou v podstatě reprezentací elementů definovaných ve značkovacích jazycích.
Viz též Element, SGML.

Příloha A

Definice záznamu o knize v jednotlivých validačních jazycích

Příklad je převzatý z [VLIST] a doplněný o definice v ostatních jazycích.

Příklad A.1. Knížka Ferdy Mravence v XML

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="8000009366">
  <title>
    Knížka Ferdy Mravence
  </title>
  <author>Ondřej Sekora</author>
  <character>
    <name>Ferda Mravenec</name>
    <friend-of>Brouk Pytlík</friend-of>
    <since>1933</since>
    <qualification>
      práce všeho druhu
    </qualification>
  </character>
  <character>
    <name>Brouk Pytlík</name>
    <since>1933</since>
    <qualification>Všechno ví, všechno zná.</qualification>
  </character>
</book>
```

Příklad A.2. Příklad definice omezení knížky v DTD

```
<!ELEMENT book (title,author,character*)>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ELEMENT title (#CDATA)>
<!ELEMENT author (#CDATA)>
```

```
<!ELEMENT character (name,friend-of?,since,qualification)>
<!ELEMENT name (#CDATA)>
<!ELEMENT friend-of (#CDATA)>
<!ELEMENT since (#CDATA)>
<!ELEMENT qualification (#CDATA)>
```

Příklad A.3. Stejná definice v DTD s předdefinovanými typy

```
<!ENTITY % isbnType "#CDATA">
<!ENTITY % titleType "#PCDATA">
<!ENTITY % authorType "#PCDATA">
<!ENTITY % nameType "#PCDATA">
<!ENTITY % friend-ofType "#PCDATA">
<!ENTITY % sinceType "#PCDATA">
<!ENTITY % qualificationType "#PCDATA">

<!ENTITY % characterType "name,friend-of?,since,qualification">
<!ENTITY % bookType "title,author,character*">

<!ELEMENT book (%bookType;)>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ELEMENT title (%titleType;)>
<!ELEMENT author (%authorType;)>

<!ELEMENT character (%characterType;)>
<!ELEMENT name (%nameType;)>
<!ELEMENT friend-of (%friend-ofType;)>
<!ELEMENT since (%sinceType;)>
<!ELEMENT qualification (%qualificationType;)>
```

Příklad A.4. Příklad definice v XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="nameType">
  <xs:restriction base="xs:string">
    <xs:maxLength value="32"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="sinceType">
  <xs:restriction base="xs:date"/>
</xs:simpleType>
<xs:simpleType name="descType">
```

```
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="isbnType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{10}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="characterType">
    <xs:sequence>
      <xs:element name="name" type="nameType"/>
      <xs:element name="friend-of" type="nameType" minOccurs="0" ▶
maxOccurs="unbounded"/>
      <xs:element name="since" type="sinceType"/>
      <xs:element name="qualification" type="descType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="bookType">
    <xs:sequence>
      <xs:element name="title" type="nameType"/>
      <xs:element name="author" type="nameType"/>
      <xs:element name="character" type="characterType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="isbnType" use="required"/>
  </xs:complexType>
  <xs:element name="book" type="bookType"/>
  </xs:schema>
```

Příklad A.5. Definice v Relax NG s použitím knihovny datových typů z XML Schema

```
  <?xml version="1.0" encoding="UTF-8"?>
  <grammar xmlns="http://relaxng.org/ns/structure/1.0" ▶
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <define name="isbnType">
    <data type="nonNegativeInteger">
      <param name="pattern">[0-9]{10}</param>
    </data>
  </define>

  <define name="titleType">
    <data type="token"/>
  </define>

  <define name="authorType">
    <data type="token"/>
  </define>
```

```
<define name="nameType">
  <data type="token">
    <param name="maxLength">32</param>
  </data>
</define>

<define name="friend-ofType">
  <data type="token"/>
</define>

<define name="sinceType">
  <data type="date"/>
</define>

<define name="qualificationType">
  <data type="token"/>
</define>

<define name="characterType">
  <element name="name">
    <ref name="nameType"/>
  </element>
  <optional>
    <element name="friend-of">
      <ref name="friend-ofType"/>
    </element>
  </optional>
  <element name="since">
    <ref name="sinceType"/>
  </element>
  <element name="qualification">
    <ref name="qualificationType"/>
  </element>
</define>

<define name="bookType">
  <attribute name="isbn">
    <ref name="isbnType"/>
  </attribute>
  <element name="title">
    <ref name="titleType"/>
  </element>
  <element name="author">
    <ref name="authorType"/>
  </element>
  <zeroOrMore>
```

```
    <element name="character">
      <ref name="characterType"/>
    </element>
  </zeroOrMore>
</define>

<start>
  <element name="book">
    <ref name="bookType"/>
  </element>
</start>

</grammar>
```

Příklad A.6. Stejná definice v Relax NG ve své čitelnější kompaktní formě

```
isbnType = xsd:nonNegativeInteger { pattern = "[0-9]{10}" }
titleType = xsd:token
authorType = xsd:token
nameType = xsd:token { maxLength = "32" }
friend-ofType = xsd:token
sinceType = xsd:date
qualificationType = xsd:token
characterType =
  element name { nameType },
  element friend-of { friend-ofType }?,
  element since { sinceType },
  element qualification { qualificationType }
bookType =
  attribute isbn { isbnType },
  element title { titleType },
  element author { authorType },
  element character { characterType }*
start = element book { bookType }
```

Příklad A.7. Příklad v DSD

```
<if><element name="book"/>
  <declare>
    <contents>
      <sequence>
        <attribute name="isbn"><stringtype ref="isbnType"/></attribute>
        <element name="title"/>
        <element name="author"/>
        <repeat min="0"><element name="character"/></repeat>
```

```
      </sequence>
    </contents>
  </declare>
</if>
<if><element name="character"/>
  <declare>
    <contents>
      <sequence>
        <attribute name="name"><stringtype ref="isbnType"/></attribute>
        <element name="name"/>
        <repeat min="0" max="1"><element name="friend-of"/></rapeat>
        <element name="since"/>
        <element name="qualification"/>
      </sequence>
    </contents>
  </declare>
</if>
<if><element name="qualification"/>
  <declare>
    <contents>
      <string/>
    </contents>
  </declare>
</if>
<if><element name="since"/>
  <declare>
    <contents>
      <stringtype ref="dateType"/>
    </contents>
  </declare>
</if>
<if>
  <or>
    <element name="friend-of"/>
    <element name="title"/>
    <element name="author"/>
  </or>
  <declare><contents>
    <stringtype ref="nameType"/>
  </contents></declare>
</if>

<stringtype id="nameType">
  <repeat min="0" max="32">
    <char/>
  </repeat>
```

```
</stringtype>

<stringtype id="isbnType">
  <repeat number="10">
    <char min="0" max="9"/>
  </repeat>
</stringtype>

<stringtype id="dateType">
  <sequence>
    <optional><repeat number="2"><stringtype ▶
ref="x:digit"/></repeat></optional>
    <optional><string value="-"/></optional>
    <optional><repeat number="2"><stringtype ▶
ref="x:digit"/></repeat></optional>
    <optional><string value="-"/></optional>
    <repeat number="4"><stringtype ref="x:digit"/></repeat>
  </sequence>
</stringtype>
```

Příklad A.8. Příklad v Schematronu

```
<pattern name="Kontrola structury">
  <rule context="book">
    <assert test="@isbn">isbn atribut je požadován.</assert>
    <assert test="(*[position()=1] = title) and (*[position()=2] = ▶
author)">
      Sekvence elementů author a title.
    </assert>
    <assert test="count(*) = 2 + count(character)">
      book nesmí obsahovat jiné elementy než title, author a charactek.
    </assert>
   <assert test="(*[position()=1] != character) and (*[position()=2] != ▶
character)">
      character následuje až po elementech title a author.
    </assert>
  </rule>
  <rule context="character">
    <assert test="*[position()=1] = name">
      První v sekvenci je name.
    </assert>
    <assert test="((*[position()=2] = friend-of) and (*[position()=3] = ▶
since) and (*[position()=4] = qualification)) or ((*[position()=2] = ▶
since) and (*[position()=3] = qualification))">
      Sekvence pro character musí obsahovat name, friend-of, since a ▶
qualification,
```

```
      nebo jen name, since a qualification.
    </assert>
  </rule>
</pattern>
<pattern name="Kontrola některých datových typů">
<!-- datum není možné zkontrolovat -->
  <rule context="book">
    <assert test="string-length(author) < 32">
      author musí být kratší než 32 znaků.
    </assert>
    <assert test="string-length(title) < 32">
      title musí být kratší než 32 znaků.
    </assert>
    <assert test="(number(@isbn) > 999999999) and (number(@isbn) < ▶
1000000000)">
      isbn musí být vyjádřeno jako 10 číslic
    </assert>
  </rule>
  <rule context="character">
    <assert test="string-length(name) < 32">
      name musí být kratší než 32 znaků.
    </assert>
  </rule>
</pattern>
```